

Lösung Übungsblatt Nr. 3 / ALP 1 zur Abgabe 19.11.2003

Aufgabe 1:

Programmcode

```
myreverse :: [a] -> [a]
myreverse [] = []
myreverse (head:tail) = myreverse tail ++ [head]
```

Testlauf

```
Main> myreverse ["a", "b", "c"]
["c", "b", "a"]
Main> myreverse ["Das", "isn", "Test"]
["Test", "isn", "Das"]
Main> myreverse [1,2,3,4,5,6,7,8,9,0]
[0,9,8,7,6,5,4,3,2,1]
```

Beschreibung

Das Programm bekommt eine Liste von beliebigem Typen und gibt sie in umgekehrter Reihenfolge wieder aus. Dazu trennt es das Erste Element der Liste ab, und hängt es an den verbleibenden Rest an, welcher zuvor rekursiv nach der gleichen Methode umgedreht wird.

Aufgabe 2:

Programmcode

```
lc_count :: String -> Int
lc_count xs = length [x | x <- xs, ((ord(x) > 96) && (ord(x)<123)) ]
```

Testlauf

```
Main> lc_count "Hallo"
4
Main> lc_count "Hallo Welt"
7
Main> lc_count "AUFGABE GESCHAFFT"
0
```

Beschreibung

Das Programm bekommt einen String als Eingabewert, und verarbeitet ihn Schritt für Schritt als Liste vom Typ Character. Dabei wird die Anzahl der Elemente gezählt, auf die der Filter "OrdinalZahl vom AktuellenBuchstaben > 96 AND OrdinalZahl vom AktuellenBuchstaben <123" zutrifft.

Aufgabe 3:

Verschiebt sich auf den 4. Aufgabenzettel.

Lösung Übungsblatt Nr. 3 / ALP 1 zur Abgabe 19.11.2003

Aufgabe 4:

Programmcode

```
create_list_of_distances :: Int -> [Float]
create_list_of_distances 0 = []
create_list_of_distances attempts = [sqrt( ((fromInt(((25173*attempts*2)+13849) `mod` 65536) / 65536)^2) +
((fromInt(((25173*(attempts*2-1))+13849) `mod` 65536) / 65536)^2) )] ++ create_list_of_distances (attempts-1)

mypi :: Int -> Float
mypi 0 = pi
mypi attempts = (fromInt((length [x | x <- (create_list_of_distances attempts), x<=1])) / (fromInt(attempts)/
4))
```

Testlauf

```
Main> mypi 4000
3.269
Main> mypi 16000
3.2675
Main> mypi 30
3.06667
```

Beschreibung

Die Funktion `create_list_of_distances` gibt eine Liste mit `attempts` Elementen aus wobei die Elemente die Entfernung zum Ursprung des Quadrates darstellen.

Die Elemente berechnen sich nach dem Satz des Pythagoras aus:
Wurzel aus:(((Zufallszahl zwischen 0 und 1) zum Quadrat) + (anderer Zufallszahl zwischen 0 und 1) zum Quadrat)

Die Funktion `mypi` ist das Hauptprogramm.
Es lässt mit Hilfe von `count_elements_of_list` die Anzahl der Elemente der aus `create_list_of_distances` entstandenen Liste auf die der Filter `<= 1` wirkt zählen.

Diese Anzahl bezeichnet die Treffer im und auf dem Viertelkreis.

Die Anzahl der Treffer dividiert durch
(die Anzahl der Versuche dividiert durch 4)
ergibt den Näherungswert von Pi.

Achtung: Für die Angabe Versuchs-Anzahl (`attempts`) = 0 gibt das Programm den Vergleichswert Pi aus !!

Warum Anzahl der Versuche nochmal dividiert durch 4? Erklärt an einem Beispiel:

In ein Quadrat mit der Seitenlänge $a=4LE$ passt genau ein Kreis mit dem Radius $r=2LE$.
(Alle restlichen Angaben auch in LE, bzw. FE)
Damit ergibt sich für den Kreis ein Flächeninhalt von $A_{Kgesamt}=Pi*r^2$, also $A_{Kgesamt}=Pi*4$
und für das Quadrat $A=a^2$ also $A_{Qgesamt}=16$.
Da wir in diesem Versuch nur ein Viertel des Kreises im Quadrat betrachten, reduzieren sich die Flächeninhalte auf jeweils $1/4$.
 $A_{K1/4}=Pi$ und $A_{Q1/4}=4$.
Das heisst: Wenn der Flächeninhalt des Viertelquadrats = 4 ist, so ist der Flächeninhalt des Viertelkreises = Pi.
Der Flächeninhalt des Quadrats wird beim Programm durch die gesamte Anzahl an Versuchen dargestellt, der Flächeninhalt des Kreises durch die Anzahl der "Treffer".
Daraus folgt:
Pi verhält sich zu 4 wie Treffer-Anzahl zu Versuchs-Anzahl
Über das Kreuzprodukt ergibt sich für Pi:
 $Pi = 4*Treffer-Anzahl / Versuchs-Anzahl$
bzw. umgestellt:
 $Pi = "Treffer"-Anzahl / (Versuchs-Anzahl / 4)$
Und nichts anderes macht das Programm.

Anmerkung: Die Qualität des Näherungswertes von Pi hängt extrem stark von der Qualität der Zufallszahlen ab. Mit dem uns zur Verfügung stehenden Zufallsgenerator lassen sich für dieses Programm nur sehr schlechte Näherungswerte zu Pi erzielen. Die von uns ausprobierte Skala erreichte teilweise Werte von $\approx 2,8$ bis $\approx 3,6$. Skalierung erfolgte per `(25173*attempts*random_factor)+13849 `mod` 65536`.

Mit einem "echten" Pseudozufallsgenerator, der die Millisekunden der Systemuhr als Grundlage der Berechnung nimmt, liessen sich wesentlich bessere Näherungswerte erreichen.