Bitte bearbeiten Sie die folgenden Aufgaben in Zweiergruppen. Wenn Sie Feedback zu Ihren Lösungen haben möchten, können Sie diese in elektronischer Form per Mail an h.g.esser@gmx.de abgeben (bitte keine Mehrfachabgabe identischer Lösungen). Es gibt keine Bewertung/Benotung.

## 1. Floating-Point-Zahlen

**a)** Betrachten Sie das folgende Programm (*gleitkomma.py*, Sie finden es auch als Datei auf der Vorlesungswebseite http://hm.hgesser.de/):

```
#!/usr/bin/env python

# Rechnerarchitektur, WS 2010/11, Hochschule München
# Hans-Georg Eßer
#
# gleitkomma.py  v 0.1  (2010/10/27)


# Nutzt Bibliotheksfunktionen
# - argv (Aufrufargumente)
# - exit (Programm mit Fehlercode beenden)
from sys import argv, exit


def DualToIEEE (vz, exp, bitstr, laenge):
  # Umwandeln in 32-/64-Bit-Gleitkommazahl nach IEEE 754 (wie Vorlesung)
  # vz: Vorzeichen (0,1)
  # exp: Exponent
  # bitstr: Bit-String
  # Ausgabe: IEEE-754-Darstellung (32 oder 64 Bit)
  if laenge == 64:
    explen = 11    # Länge des Exponenten
    manlen = 52    # Länge der Mantisse; 1 + 11 + 52 = 64
    exzess = 1023  # Exzess 1023 (2^(11-1)-1)
  elif laenge == 32:
    explen = 8     # Länge des Exponenten
    manlen = 23    # Länge der Mantisse; 1 + 8 + 23 = 32
    exzess = 127   # Exzess 127 (2^(8-1)-1)
  else:
    exit(1)

  bitstr = bitstr + laenge*"0"  # hinten mit Nullen auffüllen
  # Exponent
  e = exp + exzess              # Exzess zum Exponenten addieren
  e_str = bin(e)[2:]            # Dualdarstellung ("0b" abschneiden)
  e_str = (explen-len(e_str))*"0" + e_str  # führende Nullen
  # Mantisse
  m_str = bitstr[1:manlen+1]    # Mantisse ohne führende 1!

  return (str(vz), e_str, m_str)



def BinStrToHexStr (b):
  # wandelt Binär-String ("101010") in Hex-String um
  # Leider funktioniert  hex(int("0b"+bitstring))  nicht
  v = 0
  bits = b
  while len(bits)>0:
    v = 2 * v + int(bits[0])     # vorderstes Bit in bits[0]
    bits = bits[1:]
  hexstring = hex(v)
  hexstring = hexstring[2:]      # "0x" abschneiden
  if hexstring[-1] == "L":
    hexstring = hexstring[:-1]   # "L" (long) abschneiden
  hexstring = hexstring.upper()  # Großbuchstaben
  # Leerzeichen nach ja vier Hex-Ziffern einfügen
  h = ""
  while len(hexstring)>0:
    h = h + hexstring[:4] + " "
    hexstring = hexstring[4:]
  return h
```



```
anleitung = """\
Aufruf: gleitkomma.py ZAHL (mit Dezimalpunkt)
        z. B. gleitkomma 18.4"""

# Aufrufargument einlesen ...
try:
  argument = argv[1]
except:
  print anleitung
  exit(1)

# ... und in Zahl umwandeln
try:
  zahl = float(argument)
except:
  print anleitung
  exit(1)

# Vorzeichen bestimmen
a = zahl
if a < 0:
  vz = 1       # negativ: 1
  vzstr = "-"
else:
  vz = 0       # positiv: 0
  vzstr = "+"

a = abs(a)     # Vorzeichen weg lassen

# aufteilen in Vor- und Nachkomma-Teil
vorkomma  = int(a)
nachkomma = a - vorkomma

# Vorkommastellen in Dualdarstellung umwandeln
vorkomma_bits = ""
v = vorkomma
while v > 0:
  bit = v % 2
  v = v / 2
  vorkomma_bits = str(bit) + vorkomma_bits

# Nachkommastellen in Dualdarstellung umwandeln
nachkomma_bits = ""
n = nachkomma
counter = 52
while (n > 0) and (counter > 0):
  counter = counter - 1
  n = n * 2
  bit = int(n)     # nur Vorkomma-0
                   # oder Vorkomma-1 von n
  n = n - int(n)   # Vorkommastelle auf 0 setzen
  nachkomma_bits = nachkomma_bits + str(bit)
if counter == 0:
  mehr_stellen = True

# Dualzahl: Vorzeichen, Vorkomma-Bits,
# Nachkomma-Bits
dual = vzstr + vorkomma_bits + "." + \
       nachkomma_bits

# Ergebnisse ausgeben
print "Zahl:           ", zahl
print "Vorzeichen:     ", vz, "("+vzstr+")"
print "Vorkomma:       ", vorkomma
print "Vorkomma(b):    ", vorkomma_bits
print "Nachkomma:      ", nachkomma
print "Nachkomma(b):   ", "(0.)" + nachkomma_bits
print "Dualdarstellung:", dual


# Fixkommadarstellung:

potenz = len(vorkomma_bits)-1
bits = vorkomma_bits + nachkomma_bits

while bits[0] == "0":
  bits = bits[1:]      # führende 0 abschneiden
  potenz = potenz - 1  # Potenz um 1 erhoehen

print "Fast normiert:  ", vzstr + \
      bits[0]+","+bits[1:]+ " * 2^"+str(potenz)

print
print "IEEE 754: (Vorzeichen, Exponent, Mantisse)"

(vz_str, e_str, m_str) = \
     DualToIEEE (vz, potenz, bits, 64)
print "DualToIEEE(64): ", vz_str, e_str, m_str
print "Hexadezimal:    ", \
     BinStrToHexStr (vz_str + e_str + m_str)

(vz_str, e_str, m_str) = \
     DualToIEEE (vz, potenz, bits, 32)
print "DualToIEEE(32): ", vz_str, e_str, m_str
print "Hexadezimal:    ", \
     BinStrToHexStr (vz_str + e_str + m_str)
```

Versuchen Sie, die Funktionsweise des Programms nachzuvollziehen. Hinweise zur Umwandlung finden Sie ansatzweise in den Vorlesungsunterlagen sowie vollständig auf der Wikipedia-Seite http://de.wikipedia.org/wiki/IEEE_754.

**b)** Die 32-Bit-Darstellung nach IEEE 754 hat die folgenden Parameter:

- Vorzeichen: 1 Bit
- Exponent: 8 Bit, Exzess: 127 (= $2^{8-1} - 1$)
- Mantisse: 23 Bit

Rechnen Sie von Hand die Dezimalzahlen 255,25 und 0,2578125 in das 32-Bit-Format um und überprüfen Sie Ihre Ergebnisse mit dem obigen Python-Programm.

## 2. 32-Bit-Arithmetik

**a) Perioden.** Zahlen wie 1/3 haben im Dezimalsystem eine unendliche Darstellung als Fließkommazahl (0,3333.... oder $0,\overline{3}$). Wird eine Zahl erst ab einer bestimmten Stelle periodisch, kann man auch die Schreibweise 1/350 = 0,00**28571**428571**285714**... = $0,00\overline{285714}$ verwenden. (Der Periodenstrich läuft nur über den Ziffern, die Teil der Periode sind.)

Bei der Umwandlung von Dezimal- in Dualzahlen können auch Zahlen, deren Dezimaldarstellung „abbricht“, periodisch werden. Wir verwenden im Dualsystem dieselbe Schreibweise mit Überstreichen, also z. B. $0{,}111\overline{01} = 0{,}111\mathbf{01}01\mathbf{01}01...$).

Wandeln Sie die Dualzahlen 0,1, 0,2 und 0,3 mit Hilfe des Programms in Dualzahlen um, erkennen Sie die Periode und geben Sie die kurze Darstellung (mit Periode) an. (Hinweis: Das Programm erwartet einen Dezimalpunkt, also „0.1“ statt „0,1“.)

Beim Umwandeln der Zahlen $0{,}\overline{3}$ (0,3333...), $0{,}\overline{4}$, $0{,}\overline{5}$ usw. scheint sich auch in der Fließkommadarstellung eine gewisse Periodizität einzustellen. Probieren Sie es aus. (Sie müssen dazu Werte der Form 0.333333333333333333333333 eingeben.)

**b)** Beim Rechnen mit Fließkommazahlen gibt es ein paar Probleme:

Sei A = 9999, B = 0,0001

Wir schreiben F32(x) für die Fließkommadarstellung von x (mit 32 Bit).

A und B sind problemlos darstellbar:

- F32(A) = 461C 3C00
- F32(B) = 38D1 B717

Es gilt (A+B) – A = B, aber: F32(A+B) = F32(A), obwohl F32(B) nicht 0 ist. Woran liegt das? Wie ließe sich das Problem beheben?

## 3. Erweiterung: 16- und 128-bittig

**a)** Die neue Norm IEEE 754r (2008, `http://de.wikipedia.org/wiki/IEEE_754r`) definiert auch ein 128-bittiges Float-Format mit folgenden Parametern:

- Vorzeichen: 1 Bit
- Mantisse: 112 Bit
- Exponent: 15 Bit; mit Exzess 16383 ($= 2^{15-1} - 1$)

Außerdem ist schon in IEEE 754 ein 16-bittiges Format mit diesen Parametern definiert:

- Vorzeichen: 1 Bit
- Mantisse: 10 Bit
- Exponent: 5 Bit; mit Exzess 15 ($= 2^{5-1} - 1$)

Erweitern Sie das Programm *gleitkomma.py* so, dass es auch die 16- und 128-bittigen Darstellungen ausrechnet. Dazu müssen Sie sowohl die Funktion `DualToIEEE()` als auch die Ausgabe unten im Hauptprogramm erweitern.

**b)** Machen Sie einen Vorschlag für ein 512-bittiges Float-Format, geben Sie also die Parameter dafür an (Vorzeichen, Mantisse, Exponent, Exzess); orientieren Sie sich dabei an den Parametern der 32- bis 128-bittigen Varianten.