ALP II Objektorientierte Programmierung
SS 2012
Prof. Dr. Margarita Esponda

1. Teilklausur

Name: ..... **Vorname:**..... **Matrikel-Nummer:**.....

Ich bin mit der Veröffentlichung der Klausurergebnisse mit Matrikel-Nummer und Note im Internet einverstanden:

☐ ja ☐ nein

Unterschrift:.....

Die maximale Punktzahl ist 100.

Aufgabe	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**	**Summe**	**Note**
Max. Punkte	8	16	16	12	12	8	8	6	14		
Punkte											

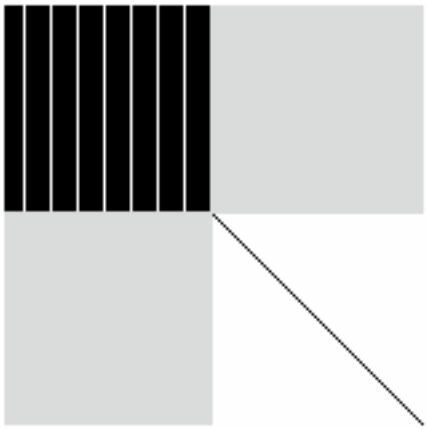
Viel Erfolg!

1. Aufgabe (8 Punkte)
Was ist der Wert und Typ folgender Python-Ausdrücke?

Ausdruck	**Wert**	**Typ**
9//2/2	**2.0**	**float**
(~3+1)<<2	**-12**	**int**
[k%2 **for** k **in** range(5)]	**[0,1,0,1,0]**	**list**
[a+a **for** a **in** "abc"]	**['aa', 'bb', 'cc']**	**list**

2. Aufgabe (16 Punkte)

Programmieren Sie nur mit Hilfe von **if-else-**Anweisungen und logischen Ausdrücken den Inhalt der Funktion **decide_color**, so dass folgendes Bild reproduziert wird.



Die **decide_color** Funktion entscheidet, welche Farbe ein Pixel an einer beliebigen **x, y** Koordinate haben soll. Die **decide_color** Funktion wird innerhalb einer **paint**-Funktion für jede **x, y** Koordinate aufgerufen, so dass ein quadratisches Bild mit Seitenlänge **size** (100 Pixels) gezeichnet wird. Nehmen Sie an, dass Sie nur folgende Farbennamen zur Verfügung haben:

['black', 'gray', 'white']

Lösung:

```
def decide_color (x, y, size):
    middle = size//2
    if x<middle and y<middle:
        if x%10==0:
            return "white"
        else:
            return "black"
    else:
        if x>middle and y>middle:
            if x==y:
                return "black"
            else:
                return "white"
        else:
            return "gray"
```

3. Aufgabe (16 Punkte)

Implementieren Sie eine möglichst effiziente iterative Python-**Funktion**, die diese Definition des Binomialkoeffizienten für die Berechnung verwendet.

$$\binom{n}{k} = \begin{cases} 1 & falls \quad k = 0 \\ \prod_{i=1}^{k} \frac{n-(i-1)}{i} & falls \quad k > 0 \end{cases}$$

$$= \begin{cases} 1 & falls \quad k = 0 \\ \frac{n}{1} \cdot \frac{(n-1)}{2} \cdot \frac{(n-2)}{3} \cdots \frac{(n-k+2)}{(k-1)} \cdot \frac{(n-k+1)}{k} & falls \quad k > 0 \end{cases}$$

Lösung:

```
def binom_iter(n, k):
    if 0<=k<=n:          # wurde bei der Korrektur nicht unbedingt verlangt
        if k==0:
            return 1
        else:
            nom, denom = 1, 1
            for i in range(1, k+1):
                nom *= (n-(i-1)
                denom *= i
            return nom//denom
    else:
        return 0
```

4. Aufgabe (12 Punkte)

Programmieren Sie eine eigene **endrekursive** Funktion in Python (natürlich ohne Verwendung des Potenz-Operators), die bei Eingabe von zwei positiven ganzen Zahlen ***a*** und ***b*** die Potenz a^b berechnet.

1. Lösung:

```
def power_rec(result,basis,exp):
    if exp>0:
        return power_rec(result*basis, basis, exp-1)
    else:
        return result
```

2. Lösung:

```
def power_rec(basis,exp):

    def help_power(result, basis, exp):
        if exp>0:
            return help_power(result*basis, basis, exp-1)
        else:
            return result

    return help_power(1, basis, exp)
```

5. Aufgabe (12 Punkte)

In der imperativen Implementierung des Quicksort-Algorithmus spielt das Umsortieren der Zahlen innerhalb der Partitionierungs-Funktion eine wesentliche Rolle, weil die Teilung der Zahlen in zwei Bereiche in dem Array selber stattfindet (sortieren „in-place").

a) Erläutern Sie einen Durchgang des **Partitionierungs**-Teils des Algorithmus aus den Vorlesungsfolien anhand folgender Zahlensequenz.

A = [5,6,2,7,1,9,0]

Lösung:

[5, 2, 6, 7, 1, 9, 0]

[5, 2, 1, 7, 6, 9, 0]

[5, 2, 1, 0, 6, 9, 7]

[0, 2, 1, 5, 6, 9, 7]

b) Ist der Quicksort-Algorithmus stabil? Begründen Sie Ihre Antwort.

Lösung:

Nein.

Beispiel:

[5, 2, 6, 7, 2]
[5, 2, 2, 7, 6]
[2, 2, 5, 7, 6] Die zweite Zahl 2 würde gegen die Zahl 5 vertauscht!

c) Es gibt eine Reihe Sortieralgorithmen, die ohne Vergleiche und mit linearem Aufwand Daten sortieren können (z.B. Countingsort). Welche Eigenschaften müssen die Daten erfüllen, damit solche linearen Algorithmen verwendet werden können? Geben Sie ein Beispiel.

Lösung:

Wenn die Daten, die sortiert werden sollen, ganzzahlige Werte mit einem kleinen Wertebereich zwischen **0** und **k** sind, ist es möglich, Zahlen zu sortieren ohne diese direkt zu vergleichen. Sinn macht es allerdings nur, wenn **k** viel kleiner als die Größe der zu sortierenden Datenmenge ist, weil sonst der zusätzliche Speicherverbrauch unnötig groß wäre.

Buchstaben, Zeitangaben, Datums, usw.

6. Aufgabe (8 Punkt)

Wenn Sie die Buchstaben des Arrays **A** mit Hilfe des **rekursiven Mergesort**-Algorithmus sortieren (der die Zahlen mit Hilfe eines Hilfsarray sortiert) und am Ende jedes Aufrufs der **merge**-Funktion das gesamte Array **A** zwischendurch ausgeben, wie würde die Ausgabesequenz aussehen?

A = [m, e, r, g, e, f, u, n]

Lösung:

[m, e, r, g, e, f, u, n]
[e, m, r, g, e, f, u, n]
[e, m, g, r, e, f, u, n]
[e, g, m, r, e, f, n, u]
[e, g, m, r, e, f, n, u]
[e, g, m, r, e, f, n, u]
[e, e, f, g, m, n, r, u]

7. Aufgabe (8 Punkt)

Erläutern Sie die **max_heapify**-Funktion aus der Vorlesung, die innerhalb des Heapsort-Algorithmus verwendet wird, um in einem beliebigen Node eines Heaps die **max_heapify**-Eigenschaft wieder herzustellen. Analysieren Sie mit Hilfe der **O**-Notation die Komplexität der Funktion.

Lösung:

Die **max_heapify**-Funktion bekommt ein Feld **H** und eine Position des H-Feldes als Parameter.

Die Funktion geht davon aus, dass das linke und rechte Kind der angegebenen Position die **max_heap**-Eigenschaft besitzen und überprüft zuerst nur, ob die heap-Eigenschaft an der angegebenen Position erfüllt wird. Wenn das der Fall ist, wird nichts getan und die Ausführung der Funktion wird beendet.

Wenn die heap-Eigenschaft nicht erfüllt wird, wird das Problem korrigiert, indem die angegebene Position mit dem größeren von beiden Kindern vertauscht wird. Dann wird die Funktion rekursiv mit dem veränderten Kind aufgerufen, weil die heap-Eigenschaft des veränderten Kinderknotens eventuell nicht mehr erfüllt wird.

Weil ein Heap immer einen balancierten Binärbaum darstellt, ist die maximale Anzahl von rekursiven Aufrufen gleich dem Abstand zwischen der Baumwurzel und einem beliebigen Blatt, was der Funktion log(n) entspricht.

D.h. die Komplexität der max_heap-Funktion ist O(log(n)).

8. Aufgabe (6 Punkte)

Gegeben sei die folgende Programmformel

$$\{P\} \equiv \{i + j == 0\}$$

i = i+1
j = j-1

$$\{Q\}$$

Mit welchen der folgenden aufgeführten Nachbedingungen erhalten wir eine gültige Programmformel? Begründen Sie Ihre Antwort.

☐ $Q \equiv i + j - 1 == 0$

☒ $Q \equiv i + j == 0$

☐ $Q \equiv i + j + 2 == 0$

☐ $Q \equiv i + j == 1$

☐ $Q \equiv i + j == 2$

Begründung:

$$\{P\} \equiv \{i + j == 0\}$$

$$\Updownarrow$$

$$\{R'\} \equiv \{i + 1 + j - 1 == 0\}$$ ..... Zuweisungsaxiom

i = i+1 ... Sequenzregel

$$\{R\} \equiv \{i + j - 1 == 0\}$$ ..... Zuweisungsaxiom

j = j-1

$$\{Q\} \equiv \{i + j == 0\}$$

9. Aufgabe (14 Punkte)

Gegeben sei folgendes Python-Programm, das die Anzahl der Bits ohne führende Nullen von einer positiven Zahl berechnet und in der Variablen z ablegt.

$$\{P\} \equiv \{Zahl \geq 0\}$$

```
hilf = Zahl
bits = 1

while hilf>1:
    hilf = hilf//2
    bits = bits+1
```

$$\{Q\} \equiv \{bits == BinaryDigits(Zahl)\}$$

Dabei sei $BinaryDigits(z)$ die Anzahl der Bits der Binärdarstellung der Zahl **z** ohne führende Nullen.

Beispiel: $BinaryDigits(6)$ ergibt 3

a) Welche der folgenden Prädikate stellen eine geeignete Invariante der **while**-Schleife dar?

- [x] $hilf \geq 0 \wedge (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)$
- [ ] $hilf \geq 0 \wedge (BinaryDigits(hilf) + 1 == BinaryDigits(Zahl) + bits)$
- [ ] $hilf \geq 0 \wedge (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + bits)$

b) Beweisen Sie die **Invarianten**-Eigenschaft für das von Ihnen unter a) genannte Prädikat. Sie können benutzen, dass für jede positive ganze Zahl ***z*** folgendes gilt:

$$BinaryDigits(z) = \begin{cases} 1, & falls \quad z = 0 \quad oder \quad z = 1 \\ BinaryDigits(z\,//\,2) + 1, & falls \quad z > 1 \end{cases} \qquad \ldots\ 1)$$

Lösung:

Zu beweisen ist:

$$INV \equiv hilf \geq 0 \wedge (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)$$

while hilf>1:
  hilf = hilf//2
  bits = bits+1

$$INV \wedge (hilf \leq 1)$$

Wir müssen dann nach der **while**-Regel zeigen, dass

$$INV \wedge (hilf > 1)$$

  hilf = hilf//2
  bits = bits+1

$$INV$$

.....

$$hilf > 1 \wedge (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)$$

aus 1) folgt:

$$hilf > 1 \wedge (BinaryDigits(hilf\ //2) + 1 + bits == BinaryDigits(Zahl) + 1)$$

$\Downarrow$ .... KR II

$$hilf \geq 0 \wedge (BinaryDigits(hilf\ //2) + bits == BinaryDigits(Zahl))$$ ... ZA

  hilf = hilf//2

$$hilf \geq 0 \wedge (BinaryDigits(hilf) + bits == BinaryDigits(Zahl))$$ ... ZA

  bits = bits+1

$$hilf \geq 0 \wedge (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)$$

.....

nach der **Sequenz**-Regel, **Konsequenz**-Regel II und der **while**-Regel gilt dann, dass $INV \equiv hilf \geq 0 \wedge (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)$

eine Invariante der **while**-Schleife ist.