1. Aufgabe: Datenstrukturen (36 Punkte)

a) **(8 Punkte)** In der Informatik werden häufig die Bezeichnungen FiFo (First in, First out) und LiFo (Last in, First out) benutzt. Geben Sie an, welche Datenstrukturen nach diesen Prinzipien arbeiten und erklären Sie ihren Aufbau.

b) **(4 Punkte)** In einen Info-LKW, dessen Ladefläche nur von hinten zugänglich ist, passen immer zwei I-Paletten nebeneinander. Begründen Sie, dass sich die Realisierung des LKWs in einer Klasse am einfachsten mit zwei Stapeln bewerkstelligen lässt.

c) **(12 Punkte)** Realisieren Sie die Klasse, indem Sie den entsprechenden Python-Code notieren. Beachten Sie dabei die Klassenbeschreibung im Anhang. Sie dürfen davon ausgehen, dass die Klasse IPalette bereits definiert ist.

d) **(12 Punkte)** Ein Mitarbeiter in einem Büro möchte zum Verwalten seiner Aufgaben ein Programm schreiben. Er möchte dazu auf eine der vier Datenstrukturen Liste, Schlange, Prioritätsschlange oder Stapel zurückgreifen. Bewerten Sie die Eignung der Datenstrukturen für diese Aufgabe.

Anhang

Dokumentation der Klasse InfoLKW

Konstruktor	**InfoLKW()** Erzeugt einen neuen LKW mit zwei leeren Stapeln.
Auftrag	**einladen(palette: IPalette, links: bool)** Fügt eine Palette dem LKW hinzu. Dabei gibt `links` an, ob dieses in den linken oder rechten Stapel geschehen soll.
Auftrag	**umstellen(anz: int, links: bool)** Lädt eine angegebene Anzahl von Paletten, falls vorhanden, beim Zutreffen des zweiten Parameters von links nach rechts. Sonst wird von rechts nach links umgestellt. Ist die Anzahl der vorhandenen Paletten kleiner, werden entsprechend weniger umgestellt.
Anfrage	**ausladen(links: bool): IPalette** Entfernt eine Palette, sofern vorhanden, aus dem linken oder rechten Stapel. Sie liefert `null`, falls keine entsprechende Palette mehr vorhanden ist.

Lösungen:

Datenstrukturen (36 Punkte):

a) (8 Punkte) Die Schlange arbeitet nach dem FiFo-Prinzip. An einem Ende werden neue Elemente hinzufügt, am anderen Ende werden die Elemente entnommen. Der Stapel arbeitet nach dem LiFo-Prinzip. Nur an einem Ende können Elemente hinzugefügt und wieder entfernt werden. Ein Vertauschen von Plätzen ist in beiden Fällen nicht möglich.

(2x 2P Zuordnung 2P Aufbau)

b) (4 Punkte) Da die linke und die rechte Seite unabhängig voneinander sind, ist jeder ein eigener Stapel. Durch die Be- und Entladung nur an einer Seite wird das Stapelprinzip realisiert.

c) (12 Punkte)

```
1  class InfoLKW:
2
3    def __init__(self):
4      self.links= Stack()
5      self.rechts= Stack()
6
7    def einladen(self, palette: IPalette, links: bool):
8      if links:
9        self.links.push(palette)
10     else:
11       self.rechts.push(palette)
12
13   def umstellen(self, anz: int, links: bool):
14     for i in range(0,anz):
15       if links:
16         self.rechts.push(self.links.top())
17         self.links.pop()
18       else:
19         self.links.push(self.rechts.top())
20         self.links.pop()
21
22   def ausladen(self, links: bool) -> IPalette:
23     if links:
24       palette= self.links.top()
25       self.links.pop()
26     else:
27       palette= self.rechts.top()
28       self.rechts.pop()
29     return palette
```

d) (12 Punkte) Der Stapel ist ungeeignet, weil bei vielen Aufgaben solche, die tiefer im Stapel liegen, sehr lange nicht zur Bearbeitung kommen. Insgesamt ist die Schlange geeignet, da alle Aufgaben in der Reihenfolge des Eingangs bearbeitet werden. Sie wird aber den Fällen nicht gerecht, in denen Aufgaben dringender bearbeitet werden müssen. Daher ist in diesem Fall die Prioritätsschlange vorzuziehen. Mit der Liste lässt sich eine solche Organisation realisieren, sie bietet nicht die entsprechenden Methoden, um die einfach die Aufgaben nach Dringlichkeit zu sortieren. (4x 3P)

Ergebnis

sehr gut	2		
gut	4		
befriedigend	3	gesamt:	10
ausreichend	1	Schnitt:	2,30
mangelhaft	0		
ungenügend	0		