Prof. Dr. Stefan Göller
Dr. Da-Jung Cho
Daniel Kernberger

Einführung in die Informatik

WS 2019/2020

Übungsblatt 4
15.11.2019 - 21.11.2019

Abgabe: Bis zum 21.11. 18:00 Uhr über moodle. Reichen Sie pro Aufgabe, die Sie bearbeitet haben, genau eine Textdatei mit dem Namen `aufgabe_i.py`, (wobei `i` die Aufgabennummer ist) ein, welche die Lösung ihrer Gruppe enthält. Beachten Sie die speziellen Dateinamen, die in Aufgabe 2 gefordert sind. Jede Aufgabe muss mittels Rekursion gelöst werden. Außerdem dürfen keine Python-Bibliotheken eingebunden werden.

Aufgabe 1 (Anzahl der Klammerausdrücke) (20 Punkte):

Schreiben Sie eine Funktion `anzahl`, die einen Parameter `n` vom Typ `int` erwartet und die Anzahl der wohlgeklammerten Ausdrücke mit `n` Klammerpaaren rekursiv berechnet. Schauen Sie sich dazu die rekursive/induktive Definition der Menge aller wohlgeklammerten Ausdrücke an. Für `n`$=3$ sollte Ihre Funktion `anzahl` den Wert 5 zurückgeben, da es fünf wohlgeklammerte Ausdrücke mit drei Klammerpaaren gibt, nämlich

$$((())),\ (())(),\ (()()),\ ()(()),\ \text{ und } ()()().$$

Beachten Sie, dass Sie lediglich die Anzahl dieser Ausdrücke zu berechnen haben, nicht die Ausdrücke selbst!

Hinweis. Bezeichne $T(n)$ die Anzahl der wohlgeklammerten Ausdrücke über n Klammerpaaren. Es gilt $T(0) = 1$. Sei nun $n > 0$. Verwenden Sie, dass es genau $T(i) \cdot T(n - i - 1)$ wohlgeklammerte Ausdrücke e über n Klammerpaaren gibt, bei dem zwischen der ersten öffnenden Klammer von e und ihrer passenden schließenden Klammer ein wohlgeklammerter Ausdruck mit i Klammerpaaren steht ($i < n$). Beachten Sie, dass es für jeden wohlgeklammerten Ausdruck mehrere mögliche i gibt.

Aufgabe 2 (von Listen von Listen von ...) (20=10+10 Punkte):

Wie in der Vorlesung besprochen, können Sie in Python Listen `L` durch den Befehl `L[:]` kopieren. Dies kopiert jedoch nur die äußere Liste `L`, kopiert jedoch nicht die Elemente von `L`, die selbst auch veränderlich sind: Folgendes Python-Programm

```
L=[1,3]
K=[L,[2]]
U=K[:]
print(U)
```

kopiert zwar die Liste `K` und liefert die Ausgabe

`[[1,3],[2]]`.

Falls Sie diesem Programm jedoch die folgenden beiden Zeilen

```
L.append(4)
print(U)
```

anhängen würden, erhielten Sie die Ausgabe

```
[[1,3],[2]]
[[1,3,4],[2]].
```

Die Unterliste `L` hat also einen Seiteneffekt (der Tiefe 2) erzeugt, da die Unterliste `[1,3]` in `K` selbst nicht kopiert wurde. Genauso können u.U. Seiteneffekte in Tiefe 3 auftreten, indem Sie eine Liste modifizieren, die in einer Liste vorkommt, die selbst wiederum in einer Liste vorkommt (und so weiter).

Eine `int`-*Superliste* ist rekursiv folgendermaßen definiert:

- Jede Liste, deren Elemente alle vom Typ `int` sind, ist eine `int`-Superliste.
- Jede Liste deren Elemente alle `int`-Superlisten sind, ist eine `int`-Superliste.

a) Schreiben Sie eine rekursive Funktion `intSuperliste` mit einem Parameter, die überprüft, ob es sich bei dem übergebenen Parameter um eine `int`-Superliste handelt, also `True` zurückgibt, falls es sich um eine `int`-Superliste handelt und sonst `False`.

b) Schreiben Sie eine rekursive Funktion `Kopie`, die eine `int`-Superliste als Parameter erwartet und eine Kopie der `int`-Superliste zurückgibt, bei der die oben beschriebenen Seiteneffekte ausbleiben, d.h. es gibt keine Seiteneffekte beliebiger Tiefe.

Achtung: Für diese Aufgabe dürfen Sie nur Python-Befehle verwenden, welche Sie in der Vorlesung kennen gelernt haben. Insbesondere ist die Verwendung des Befehls `deepcopy` nicht erlaubt!

Hinweis: Falls `s` eine Variable ist, können Sie mittels **`type(s)==list`** (bzw. **`type(s)==int`**) testen, ob `s` tatsächlich vom Typ `list` (bzw. vom Typ `int`) ist.

Aufgabe 3 (Rekursive Funktionen) (20 Punkte):

Bitte lesen Sie sich diese Aufgabe genau durch. Die Försterin vom Habichtswald möchte die Höhe des kleinsten Baums in ihrem Wald berechnen. Sie kennt bereits die Höhe des größten Baums und weiß, dass folgende Regeln für den Habichtswald gelten:

- Wenn ein Baum der Höhe x existiert, dann auch einer der Höhe $x - 34$.
- Wenn ein Baum der Höhe x existiert, dann auch einer der Höhe $\frac{x-11}{2}$.

Schreiben Sie ein Programm, welches die Höhe des größten Baums im Habichtswald, gegeben als positive `float`-Zahl, als Eingabe erwartet und die Höhe des kleinsten Baums ausgibt. Nutzen Sie dafür eine Funktion, die rekursiv die kleinste Höhe berechnet. Beachten Sie, dass natürlich **keine Bäume der Höhe** ≤ 0 existieren.

Hinweis: Beachten Sie, dass Sie immer die Resultate beider Regelanwendungen untersuchen müssen. Für den Eingabewert 34.5 liefert eine Anwendung der ersten Regel einen Baum der Höhe 0.5. Der kleinste Baum im Habichtswald hat für den Startwert 34.5 aber die Höhe 0.375: Da wir wissen, dass es einen Baum der Höhe 34.5 gibt, ergibt sich mit der zweiten Regel, dass es auch einen Baum der Höhe $\frac{34.5-11}{2}$ gibt, also einen Baum der Höhe 11.75. Die erste Regel ist auf diesen Baum natürlich nicht mehr anwendbar. Eine weitere Anwendung der zweiten Regel liefert dann den Baum der Höhe $\frac{11.75-11}{2} = 0.375$, auf den keine Regel mehr anwendbar ist.