Informatik I: Einführung in die Programmierung

Prof. Dr. Bernhard Nebel — Universität Freiburg
Dr. Christian Becker-Asano, Dr. Stefan Wölfl — Institut für Informatik
Wintersemester 2013/2014

Übungsblatt 4

Abgabe: Freitag, 22. November 2013, 18:00 Uhr

In diesem Übungsblatt wird es darum gehen, eine Zeichenkette (Text) einzulesen und dabei eine Datenstruktur anzulegen, die es später erlaubt für ein gegebenes Wort zu entscheiden, ob und wie oft dieses Wort in dem Text vorkommt. Unter einem *Wort* verstehen wir auf diesem Übungsblatt jede endliche Aufeinanderfolge von Buchstaben des deutschen Alphabets (also den Zeichen a, . . . , z, A, . . . , Z, ä, Ä, ö, Ö, ü, Ü, ß) mit Ausnahme der leeren Zeichenfolge. Das Ende eines Wortes wird in einem Text durch (mindestens) ein nachfolgendes Leerzeichen, ein Satzzeichen („.“, „,“, „;“, „:“, „!“, „?“) oder eine neue Zeile angezeigt. All die hier gelisteten Zeichen bezeichnen wir als *legale Zeichen*.

Bei diesem Übungsblatt erwarten wir alle Lösungen zu den Aufgaben in der Datei `words.py` im Unterverzeichnis `sheet04` (beachten Sie die Formatierungshinweise für Python-Dateien, wie auf dem letzten Übungsblatt angegeben). Teilen Sie die einzelnen Aufgaben durch eine Kommentarzeile voneinander ab, etwa so:

```
## Exercise 4.1
```

In der auf der Webseite der Vorlesung verfügbaren Datei `words_data.py` finden Sie kleine Beispieltexte, die Sie für Ihre Tests verwenden können. Fügen Sie diese Datei bitte zu Ihrem SVN-Unterverzeichnis hinzu und ergänzen Sie diese um weitere Texte Ihrer Wahl. Eine Quelle ist z.B. `http://www.gutenberg.org/` (ggfs. müssen diese Texte etwas „aufbereitet“ werden).
In der Datei `words.py` können Sie die Datendatei importieren, z.B. mit:

```
from words_data import loremipsum
```

Vergessen Sie nicht, alle von Ihnen verwendeten Dateien ins SVN-Repository hochzuladen!

Aufgabe 4.1 (Alphabetische Ordnung; 4 Punkte)

Definieren Sie eine Funktion `alph_lt(x, y)` (ggf. unter Zuhilfenahme von Hilfsfunktionen), die bei Eingabe zweier Wörter `x` und `y` den Wert `True` zurückgibt, falls das Wort `x` in alphabetischer Sortierung dem Wort `y` vorhergeht; und andernfalls den Wert `False`.

Testen Sie Ihre Funktion an einer ausreichenden Anzahl von geeigneten, selbst gewählten Beispielen.

Hinweis: Bei der alphabetischen Sortierung von Wörtern werden zunächst Großbuchstaben wie Kleinbuchstaben behandelt und Umlaute wie die entsprechenden

Buchstaben ohne diakretische Zeichen (also „Ä“ wie „a“) und „ß“ wie „ss“. Beginnt das Wort `y` (unter Beachtung der bisherigen Regeln) mit dem kürzeren Wort `x`, so wird `x` vor `y` einsortiert. Ferner treffen wir für den Fall, dass die Wörter im Sinne dieser Regeln „gleich“ sind, folgende Festlegung: dasjenige Wort wird vor dem anderen Wort einsortiert, das als erstes (die Wörter von links nach rechts gelesen) den Vergleich der Buchstaben analog folgendem Schema gewinnt: „U“ vor „u“, letzteres vor „Ü“, letzteres vor „ü“. Zum Beispiel hätte man folgende Sortierung: „KUhle“ vor „KuHLe“ vor „KuHle“ vor „Kuhle“ vor „KÜhle“ vor „Kühle“.

Aufgabe 4.2 (Nächstes Wort; 4)

Definieren Sie eine Funktion `next_word(s)`, die angewendet auf einen String `s` ein Tupel `(word, rest)` zurückgibt. Dabei sei `word` das erste (vollständige) Wort von `s` und `rest` die Zeichenfolge, die in `s` auf `word` folgt.
Falls Ihre Prozedur bei der Abarbeitung von `s` auf ein nicht-legales Zeichen stößt, soll die Funktion das zuletzt gelesene Wort und den leeren String (als Rest) zurückgeben.

Testen Sie Ihre Funktion an einer ausreichenden Anzahl von geeigneten und selbst gewählten Beispielen.

Aufgabe 4.3 (Wort-Liste; 2 + 2)

Mit Hilfe der Funktion `next_word(s)` sollen Sie im Folgenden eine Liste von Wörtern erstellen, die in einem gegebenen Text `s` vorkommen. Die Liste bestehe aus Tupeln `(wrd, n)`, wo `wrd` ein in `s` vorkommendes Wort und `n` die Anzahl ist, wie oft dieses Wort vorkommt.

(a) Definieren Sie eine Funktion `word_list(s)`, die die oben angegebene Wort-Liste aus einem gegebenen Text `s` zurückgibt. Es sollen nur die Wortvorkommnisse bis zum ersten nicht-legalen Zeichen gezählt werden.

(b) Definieren Sie eine Funktion `freq_word_list(wrd, lst)`, die für ein Wort `wrd` und eine Wort-Liste `lst`, die in `lst` hinterlegte Anzahl der Wortvorkommnisse von `wrd` zurückgibt. Falls das Wort in der Liste nicht vorkommt, soll der Wert `0` zurückgegeben werden.

Testen Sie Ihre Funktionen an jeweils mindestens vier geeigneten und selbst gewählten Beispielen. Benutzen Sie in (b) zum Testen auch lange Texte (mit mehr als 1000 Wörtern).

Aufgabe 4.4 (Wort-Baum; 2 + 2 + 2)

Mit Hilfe der Funktion `next_word(s)` sollen Sie im Folgenden einen Baum der in einem gegebenen Text `s` vorkommenden Wörter wie folgt erstellen: Ein Baum ist ein 4-Tupel `(wrd, ltr, rtr, n)`, bestehend aus einem Wort `wrd`, einem Baum `ltr` (*linker Teilbaum*), einem Baum `rtr` (*rechter Teilbaum*),

und der Anzahl `n` der Vorkommnisse von `wrd` in `s`. Hierbei sollen in `ltr` nur Wörter vorkommen, die dem Wort `wrd` in Python's lexikographischer Ordnung vorhergehen, und in `rtr` nur solche Wörter, die in dieser Ordnung hinter das Wort `wrd` sortiert werden. Wo sinnvoll ist für `ltr` bzw. `rtr` auch der Wert `None` (*leerer Baum*) zugelassen.

(a) Definieren Sie eine Funktion `word_tree(s)`, die den Baum der in einem gegebenen Text `s` vorkommenden Wörter zurückgibt. Es sollen wieder nur die Wortvorkommnisse bis zum ersten nicht-legalen Zeichen gezählt werden.

(b) Definieren Sie eine Funktion `freq_word_tree(wrd, tree)`, die für ein Wort `wrd` und einen Wort-Baum `tree`, die in `tree` hinterlegte Anzahl der Wortvorkommnisse von `wrd` zurückgibt. Falls das Wort in dem Baum nicht vorkommt, soll der Wert `0` zurückgegeben werden.

(c) Definieren Sie eine Funktion `print_tree(tree)`, die alle in `tree` vorkommenden Wörter und die in `tree` hinterlegte Anzahl der jeweiligen Wortvorkommnisse zeilenweise (pro Zeile ein Wort) ausgibt. Dabei soll für jeden Knoten des Baumes folgende Regel eingehalten werden: jedes Wort, das im rechten Teilbaum von `(wrd, ltr, rtr, n)` vorkommt, wird vor dem Wort `wrd` ausgegeben und das Wort `wrd` wird vor jedem Wort im linken Teilbaum ausgegeben.

Testen Sie Ihre Funktionen jeweils an mindestens vier geeigneten und selbst gewählten Beispielen. Benutzen Sie in (b) zum Testen auch lange Texte (mehr als 1000 Wörter).

Für die Fleißigen: Verwenden Sie anstelle der lexikographischen Ordnung auch die alphabetische Ordnung aus Aufgabe 4.1.

Aufgabe 4.5 (Erfahrungen; 2 Punkte)

Legen Sie im Unterverzeichnis `sheet04` eine Textdatei `erfahrung.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, benötigter Zeitaufwand nach Teilaufgabe, Bezug zur Vorlesung, Interessantes, etc.).