Übungen zu Computergrundlagen WS 2012/2013

Übungsblatt 13: C

25. 1. 2013

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist **Freitag, 1. 2. 2013, 13:30**.
- Abgabe wie immer als Email an Deinen Tutor.
- Bitte nur C-Programmcode und eine Textdatei mit der Antwort auf die Verständnisfrage 13.2.4 schicken, *keine Binärprogramme*. Vergiss nicht, wie immer Deinen Namen als Kommentar an den Anfang der Dateien zu schreiben.

Aufgabe 13.1: Statistische Auswertung von Datenreihen (5 Punkte)

Im Verzeichnis `~arnolda/computergrundlagen/13` liegt die Datei `data.dat`, die in drei Spalten Messdaten einer Simulation enthält. In demselben Verzeichnis findest Du auch das Gerüst eines C-Programms `stats.c`, das Du so ergänzen sollst, dass es die Datenreihen einliest, Mittelwerte, Minima und Maxima der einzelnen Datenreihen berechnet und dann zusammen mit ihrem Namen ausgibt (der Name der Datenreihe steht in der ersten Zeile von `data.dat`). Gehe dabei wie folgt vor:

13.1.1: Definiere die Datenstruktur **`struct`** `datacol` zum Speichern der Daten. Es soll den Namen als **`char`**`[20]` und die eigentlichen Daten der Reihe als ein Zeiger auf **`float`** (**`float`**$*$ enthalten. (1 Punkt)

13.1.2: Schreibe eine Funktion `read_data()`, die die Daten von der Standardeingabe mit Hilfe der Funktion `scanf()` in globale Variablen mit obigen Datenstrukturen einliest. Die Datenreihen selbst sollen dynamisch, d.h. mit `malloc()` bzw. `realloc()` im Speicher angelegt werden, damit Datenreihen beliebiger Länge damit verarbeitet werden können. (2 Punkte)

Hinweis: Benutze `realloc()` so, dass Du bei jedem Einlesen eines neuen Wertes das Array um eine Fließkommazahl erweiterst. Die Funktion `scanf()` liefert die Anzahl der korrekt eingelesenen Werte zurück. Ist die Datei zu Ende, liefert sie -1 zurück. Daher kannst Du einfach überprüpfen, ob `scanf()` einen Wert > 0 zurückliefert, um zu erfahren, ob die Datei zu Ende ist.

13.1.3: Schreibe nun die drei Funktionen `compute_min(`**`float`** `*col)`, `compute_max(`**`float`** `*col)` und `compute_mean(`**`float`** `*col)`, die für jeweils eine Datenreihe den Mittelwert, das Minimum und das Maximum berechnen. (2 Punkte)

Denke daran, Dein Programm mit Hilfe der Datendatei zu testen! Schau Dir die Daten in Gnuplot an, um zu sehen, ob die ermittelten Werte sinnvoll sind. Gib dann das fertige Programm ab.

Aufgabe 13.2: Self-Avoiding Walks (5 Punkte)

Im Verzeichnis `~arnolda/computergrundlagen/13` findest Du schließlich noch die Datei `saw.c`, die das Gerüst für eine C-Implementation des Self-Avoiding Walk darstellt (vergleiche Blatt 8). Diese sollst Du nun ebenfalls zu einer vollständigen Self-Avoiding Walk-Implementation ergänzen.

13.2.1: Schreibe die Funktion `move(struct position *p)`. Diese soll die durch den *Zeiger* `p` gegebene Position zufällig in einer Richtung um ± 1 verändern, genau wie bei der Random Walk-Aufgabe. (1 Punkt)

Hinweis: Benutze dabei die Befehle zum Generieren von Zufallszahlen vom letzten Übungsblatt! Beachte, dass es in C keinen direkten Weg gibt, um ganzzahlige Zufallszahlen zwischen vorgegebenen Werten zu erzeugen. Die einfachste, korrekte Methode ist, eine Fließkommazufallszahl zwischen 0 und 1 zu erzeugen, diese geeignet zu skalieren und wieder in eine Ganzzahl zu verwandeln.

13.2.2: Schreibe die Funktion `int point_in_list(struct position p, struct position *l, int len)`. Diese soll bei gegebener Liste `l` der Länge `len` von Punkten überprüfen, ob sich der Punkt `p` in der Liste `l` befindet oder nicht. Ist der Punkt enthalten, soll die Funktion 1 zurückgeben, andernfalls 0. (2 Punkte)

13.2.3: Schreibe nun die Funktion `float distance(struct position a, struct position b)`, die die quadratische Distanz $(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2$ zwischen den beiden Punkten `a` und `b` berechnet. (1 Punkt)

13.2.4: Warum ist der Programmcode in C so viel länger als der vergleichbare Python-Code? Ist die Implementation in C den Aufwand wert, wenn man dadurch wirklich längere Self-Avoiding Walks erzeugen möchte? (1 Punkt)

Hinweis: Um bessere Aussagen über das Skalierungsverhalten des Self-avoiding Walks treffen zu können, bringt es nichts, ein paar Teilchen längere Pfade zu erzeugen. Dafür sollten die Pfade wenigstens die doppelte, besser zehnfache Länge haben.