Imperative Programmierung

Aufgabenblatt X

1. Um die Personen zu speichern, wird zuerst ein neuer Typ deklariert, der aus zwei „char“-Arrays, die jeweils eine Länge von 20 haben, einer Ganzzahl vom Typ „long“ und einer Ganzzahl vom Typ „int“. Die zwei Arrays repräsentieren den Vor- bzw. Nachnamen, die erste Zahl die Matrikelnummer und die zweite Zahl das Geburtsjahr. Dann kann einfach eine Funktion gestartet werden, die zuerst eine Person in eine neue Variable einliest und dann diese Funktion rekursiv nochmal startet. Wenn keine neue Zeile mehr verfügbar ist, wird bei jeder Funktion der Rest ausgeführt, der die Personen wieder ausgibt. Dabei wird ein Effekt der Rekursion genutzt, wodurch die Personen eben in umgekehrter Reihenfolge ausgegeben werden.
   Das Programm wurde in „sortPersons.c“ implementiert.
2. Um das Array zu sortieren, wurde eine Funktion „sort“ geschrieben, die so lange die Funktion „shuffle“ ausführt, bis das Array sortiert ist. Um zu schauen, ob das Array sortiert ist, wird die Funktion „isSorted“ genutzt. Beim Sortieren wird außerdem die Anzahl an „Shuffles“ gespeichert und dann der Durchschnitt über viele Durchläufe gebildet.
   Wenn man sich dies für n=2 bis n=10 anschaut, dann ergeben sich folgende Werte, die sehr den Werten von n! ähneln:

n	shuffles	n!	Abweichung in %	Formel für Abweichung
2	0	2	100,00%	=ABS(E8-F8)/F8
3	5	6	16,67%	=ABS(E9-F9)/F9
4	22	24	8,33%	=ABS(E10-F10)/F10
5	118	120	1,67%	=ABS(E11-F11)/F11
6	759	720	5,42%	=ABS(E12-F12)/F12
7	5232	5040	3,81%	=ABS(E13-F13)/F13
8	40627	40320	0,76%	=ABS(E14-F14)/F14
9	387960	362880	6,91%	=ABS(E15-F15)/F15
10	3665669	3628800	1,02%	=ABS(E16-F16)/F16

   Daher ergibt sich eine Laufzeit O(n!). Allerdings kann die Laufzeit im Besten Fall, sogar O(1) sein, wenn die erste Permutation schon das sortierte Array ist.
   Das Programm wurde in „shuffleSort.c“ implementiert.