Dr. Johannes Mayer
Axel Blumenstock
Christian Ehrhardt
Abteilung Angewandte Informationsverarbeitung

21. Dezember 2005
Blatt 9

Allgemeine Informatik III / Systemnahe Software I (WS 2005/2006)

Abgabetermin: 10. Januar 2006

Weihnachtsmanns Liste (10 Punkte)

Natürlich hat auch beim Weihnachtsmann die moderne Technik Einzug gehalten, und er hat sein dickes Buch durch eine Textdatei ersetzt (immerhin).
Um für seine Geschenkeauswahl einen Überblick über die Altersstruktur innerhalb der Beschenkten zu gewinnen, braucht er nun ein kleines Programm, das ihm diese Datei einliest, wahlweise nach Name oder Alter auf- oder absteigend sortiert und wieder (im selben Format) ausgibt.
In dieser Textdatei steht jede Zeile für eine Person. Jede Zeile hat das Format

<*name*>`:`<*alter*>

wobei die Felder für Name und Alter keinen Doppelpunkt enthalten.
Das Verhalten des Programms soll durch Kommandozeilenoptionen gesteuert werden können. Jede Option besteht aus `'-'` vor *einem* weiteren Zeichen, plus – falls sinnvoll – einem Argument, das durch Whitespace getrennt sein kann, aber nicht muss.
Lassen Sie beim Angebot von Optionen Ihrer Kreativität freien Lauf[1], mindestens aber soll der Benutzer per Option eine Ein- und eine Ausgabedatei angeben können, sowie einen Index für die Spalte, nach der sortiert werden soll, sowie ein Flag, das die Sortierrichtung umkehrt.
Ein Aufruf des fertigen Programms könnte wie folgt aussehen (absteigendes Sortieren nach Alter):

```
$ mydb -i db.txt -s 1 -r
Elvira Erbse:92
Adam Apfel:69
Gerhard Gurke:53
Berta Bohne:48
Fridolin Fenchel:44
Karl Knoblauch:43
Hannelore Hanf:35
Diana Dattel:28
Caroline Chicoree:17
Ingo Ingwer:12
```

Bei falscher Verwendung der Optionen soll das Programm (wie üblich) seine Benutzung beschreiben und entweder abbrechen oder Defaultwerte verwenden, beispielsweise so:

```
$ mydb db.txt
ignoring non-option argument "db.txt"
Usage: mydb [ -i INFILE ] [ -o OUTFILE ] [ -s {0|1} ] [ -r ]
  -i FILE ... input file (default stdin)
  -o FILE ... output file (default stdout)
  -s N    ... sort by column N
  -r      ... reverse sort order

$ mydb -s 3
invalid -s argument: "3"
Usage: mydb [ -i INFILE ] [ -o OUTFILE ] [ -s {0|1} ] [ -r ]
  -i FILE ... input file (default stdin)
  -o FILE ... output file (default stdout)
  -s N    ... sort by column N
  -r      ... reverse sort order
```

Teilen Sie ferner den Code auf mehrere Quelltextdateien auf. Hier bietet sich an, die `main()` mit der Verarbeitung der Kommandozeilenoptionen in einer Datei wie `mydb.c` abzulegen und das Einlesen, Ausgeben und Sortieren der Daten in beispielsweise `arraylist.c`. Das Interface in Form von Funktionsdekarationen kommt in eine entsprechende Headerdatei, die mit `#include "arraylist.h"` in beiden `.c`-Quelltexten eingebunden wird.
Als Funktionen in diesem Interface bieten sich an[2]:

```
List *readList(FILE *stream);
void writeList(FILE *stream, List *list);
void sortList(List *list, int column, int reverse);
void freeList(List *list);
```

Hierbei wurde `List` per `typedef` als eine Struktur definiert, die neben einem dynamischen Array auch gleich dessen Kapazität und Zahl aktuell genutzter Einträge enthält.
Übersetzt werden kann das Ganze beispielsweise mit

```
$ gcc -Wall mydb.c arraylist.c
```

Sie können aber auch schon (im Vorgriff auf Skript Kapitel 13.4) ein Makefile erstellen, wodurch sich die nötige Kommandozeile reduziert auf ein simples

```
$ make
```

[1] Weitere denkbare Optionen könnten umfassen: eine einzelne Datei für Ein- *und* Ausgabe, Hinzufügen und Löschen von Einträgen, einen Zeilenfilter (`SELECT WHERE`), …

[2] Diese Signaturen sind Vorschläge, an die sich nicht sklavisch gehalten werden muss!

Einige nützliche Bibliotheksfunktionen[3]:

- `void *realloc (void *pointer, int newSize)` vergrößert oder verkleinert einen zuvor mit `malloc()` oder `calloc()` angelegten Speicherbereich auf die gegebene Zahl von Bytes. Der Inhalt des Speichers bleibt dabei bis zur kleineren der alten und der neuen Größe erhalten. Da der neue Speicherbereich woanders liegen kann, darf der Zeiger auf den alten danach nicht mehr verwendet werden, sondern nur noch der (zurückgegebene) Zeiger auf den neuen.

- `void qsort(void *array, int numElements, int sizeOfElement, int (*compareFunction) (const void *, const void *))` sortiert das gegebene Array mit dem Quicksort-Algorithmus. Die Zahl der Elemente im Array und die Elementgröße in Bytes ist als zweiter respektive dritter Parameter anzugeben. Der vierte Parameter ist ein Zeiger auf eine Funktion, die Zeiger auf zwei Elemente erhält und entscheidet, welches davon den höheren Rang haben soll (und damit weiter nach hinten sortiert wird). Die Bedeutung des Rückgabewerts entspricht der von `strcmp()` (vgl. letztes Blatt): Bei einem Wert größer null ist das erste Element größer als das zweite, bei kleiner null umgekehrt, und bei 0 haben sie gleichen Rang.

Viel Erfolg!

[3] Die Signaturen sind nicht ganz korrekt wiedergegeben, um die Liste übersichtlicher zu halten.