AIK Programmieren 1 Übung: Strukturen, `qsort`-Funktion

Klaus Kusche

Text sortieren mit Zeilennummern

Schreib ein Programm, das *zeilenweise* Text vom Terminal einliest (Details unten!),
jede Zeile zusammen mit ihrer *Zeilennummer* (ab 1, nicht ab 0 !) in einem Array *speichert*,
nach dem Einlesen der letzten Zeile das ganze Array *sortiert* (diesmal mit der *eingebauten Sortierfunktion* **`qsort`**, siehe unten) und die sortierten Zeilen wieder zeilenweise *ausgibt*,
wobei vor jeder Zeile ihre *ursprüngliche Zeilennummer* stehen soll.

Das Programm kannst du am besten testen, indem du es im DOS-Fenster startest
und dabei mit **<** einen Textfile in die Eingabe des Programms umleitest:

`textsort < test.txt`

Hinweise:

- Du wirst ein *Array von Strukturen* brauchen:
  Jede einzelne Struktur enthält *eine Zeile Text* (ein **`char`**-Array)
  und die dazugehörige *Zeilennummer* (als **`int`**, nicht als Text).

  Die *maximale Zeilenlänge* (Größe des **`char`**-Arrays) und die *maximale Zeilenanzahl* (Größe des **`struct`**-Arrays) darfst du fix vorgeben, mach *schöne Konstanten* dafür!
- Das Hauptprogramm besteht aus *drei* Schritten:
  - Dem *zeilenweisen Einlesen* des Textes in das Array:
    Lies in einer *Schleife* (*mitzählen!*) mit **`fgets`** (*siehe unten*) jeweils die *nächste Zeile* der Eingabe in das **`char`**-Array der *nächsten Struktur* im **`struct`**-Array
    und speichere auch die aktuelle *Zeilennummer* im **`int`** dieser Struktur.
  - Dem *Sortieren* des **`struct`**-Arrays (**`qsort`**-Aufruf, *siehe unten!*).
  - Dem *zeilenweisen Ausgeben* des sortierten Arrays:
    Schreib eine *Schleife* über alle *belegten* Elemente des Arrays
    und gib mit **`printf`** Zeilennummer und Text aus jedem Element aus.
- Mit **`scanf`** kann man Text *nicht* zeilenweise verarbeiten, denn es liest wortweise und ignoriert alle Zwischenräume, Tabs, Zeilenvorschübe usw..

  Verwende stattdessen **`fgets(`*`line`*`, `*`maxlen`*`, stdin)`** für die Eingabe.

  `fgets` liest *eine Zeile* in das **`char`**-Array ***line*** ein (setz dafür das **`char`**-Array-Member der **`i`**-ten Struktur unseres Arrays ein!), wobei das Array ***maxlen*** Zeichen groß ist (***maxlen*** ist ein **`int`**, und es werden maximal ***maxlen-1*** Zeichen gelesen).
  `stdin` steht für das Terminal (**`fgets`** könnte auch aus einer Datei lesen).

  Das **`fgets`** speichert auch das **`\n`** am Ende der Zeile im Array ***line***,
  du brauchst also beim Ausgeben der sortierten Zeilen kein **`\n`** mehr anhängen.
  Wenn eine Zeile zu lang war, enthält ***line*** *kein* **`\n`** . Prüfe das nach jedem **`fgets`**;
  gib eine Fehlermeldung aus und beende das Programm, wenn das **`\n`** fehlt.

  Achtung: Editiere deine Testdaten-Datei so, dass auch die *letzte Zeile* mit **`\n`** endet,
  denn unter Windows werden Texte meist *ohne* **`\n`** in der letzten Zeile gespeichert!

Als *Returnwert* liefert **fgets** einen Pointer auf ***line*** oder den **NULL**-Pointer, wenn der Input zu Ende ist. In diesem Fall muss deine Lese-Schleife *enden*.

Auch wenn unser **struct**-Array *voll* ist (*maximale Zeilenzahl* erreicht), muss deine Schleife *aufhören* zu lesen (gib eine *Fehlermeldung* aus!)!

- Versuche, zuerst das Hauptprogramm nur mit Einlesen und Ausgeben (ohne Sortieren) zum Laufen zu bringen.
  Bau das Sortieren erst dazu, nachdem das klappt.
- Für das Sortieren verwenden wir diesmal die *vordefinierte Funktion* **qsort** aus **stdlib.h** . Diese Funktion wird mit 4 Argumenten aufgerufen:
  - Dem zu sortierenden *Array* (unserem Array von Strukturen).
  - Der *Anzahl* der (belegten!) Elemente im Array, d.h. der Anzahl der eingelesenen Zeilen.
  - Der *Größe eines Elementes* (also der Größe unserer Struktur) in *Bytes* (was verwendest du, um diese Größe herauszubekommen?).
  - Der zu verwendenden *Vergleichsfunktion* (*nur der Name* der Funktion, *ohne* **( )** dahinter, denn die Funktion wird an dieser Stelle ja nicht aufgerufen, sondern dem **qsort** wird ein *Pointer auf die Funktion übergeben*).

  qsort kennt das „Innenleben" der Array-Elemente nicht, für **qsort** ist jedes Element einfach eine fixe Anzahl von Bytes. **qsort** weiß daher auch nicht, wie man zwei Elemente vergleicht, sondern ruft dafür die Vergleichsfunktion auf, die man ihm als 4. Argument übergibt.

  Diese *Vergleichsfunktion müssen wir selbst schreiben*.
  Sie muss wie folgt aussehen und funktionieren:
  - *Als Parameter hinein* gehen zwei *Pointer auf die zu vergleichenden Elemente*. Diese müssen beide als „**const void** *" deklariert werden, also als Pointer auf Daten unbekannten Typs, die nicht verändert werden dürfen.
  - *Als Returnwert herauskommen* muss bei der Vergleichsfunktion dasselbe wie bei **strcmp**: Ein **int**, und zwar eine Zahl **<0**, wenn das erste Argument das kleinere ist, eine Zahl **>0**, wenn das zweite Argument das kleinere ist, und **0**, wenn beide Elemente betreffend Sortierung gleichgroß sind.
  - Also müssen wir die beiden Pointer-Parameter intern auf zwei konstante *Pointer auf unseren Strukturtyp umwandeln*, und von den beiden Strukturen, auf die diese Pointer zeigen, die *Text-Member* mit **strcmp** vergleichen. Das Ergebnis von **strcmp** wird als Returnwert unserer Vergleichsfunktion zurückgegeben.

Anmerkung für die Interessierten:
Bei den Studenten 1. Sem. gibt es als Übung 9 und Übung 11 dasselbe Beispiel mit *Datei-Ein- und Ausgabe*, für *beliebig viele Zeilen* (max. Zeilenzahl nicht als Konstante vorgegeben) und mit *dynamischer Speicherung der Zeilentexte* je nach *tatsächlicher Länge* der Zeile (es wird nicht für jede Zeile die Maximallänge reserviert).