Programmieren C: Strukturen/verkettete Listen, `malloc`, File-I/O: Zeilen nach Länge sortiert ausgeben

Klaus Kusche

Gesucht ist ein C-Programm, das

- mit *einem oder mehreren Dateinamen* auf der Befehlszeile aufgerufen wird,
- alle diese Text-Dateien der Reihe nach *zeilenweise einliest*,
- und am Ende den Inhalt aller Dateien wieder *zeilenweise ausgibt*, und zwar
  - mit *Dateinamen* und ursprünglicher *Zeilennummer* vor jeder Zeile,
  - mit den Zeilen *sortiert nach deren Zeilenlänge* (die *kürzesten zuerst*),
  - und bei *gleich langen* Zeilen *in der Reihenfolge des Einlesens*.

Wir wollen das Problem ***nicht*** mit **`qsort`** lösen.
Erstens wäre das ineffizienter als die hier beschriebene Lösung,
und zweitens wäre der letzte Punkt nicht erfüllt:
`qsort` erhält die Reihenfolge gleich großer Elemente beim Sortieren *nicht*.

Unser Programm soll *beliebig viele Files mit beliebig vielen Zeilen* verarbeiten können.
Die einzelnen *Zeilen* sollen also nicht in einem fix dimensionierten Array gespeichert sondern *dynamisch angelegt* werden.
Auch die *Zeilentexte* sollen *dynamisch* in ihrer tatsächlichen Größe gespeichert werden.

Grundidee:

- Wir legen eine *maximale Länge für die Eingabezeilen* fest.
- Wir legen ein *Array von Listen* an, das so groß ist wie diese maximale Länge.
- In der *i-ten Liste* in diesem Array werden *alle Zeilen der Länge i* gespeichert, wobei neue Zeilen *hinten* an die jeweilige Liste angehängt werden.
  Für unsere Zwecke reichen *einfach verkettete Listen*.

Im Detail:

- Definiere eine *Konstante für die maximale Zeilenlänge*.
- Deklariere einen *Strukturtyp für ein Listenelement*, d.h. für *eine Zeile*.
  Er enthält einen Pointer auf den *Filenamen*, die *Zeilennummer*,
  einen Pointer auf den *Text der Zeile*, und einen Pointer für die *Listen-Verkettung*.
- Deklariere dann einen zweiten *Strukturtyp für eine Liste als Ganzes*.
  Er enthält zwei Pointer auf Listen-Elemente: *Head und Tail*
  (da wir an die Listen *effizient hinten anhängen* wollen, müssen unsere Listen nicht nur einen Head-Pointer, sondern *auch einen Tail-Pointer* haben).
- Dann kommt die *Funktion mit der eigentlichen Listen-Logik*:
  Sie soll eine *einzelne Zeile* dynamisch *speichern* und *an die richtige Liste anhängen*.

  Die Funktion hat *4 Parameter*, aber *keinen Returnwert*:
  - Unser *Array* von Listen-Strukturen.
  - Den *Filenamen* und die *Zeilennummer* der Zeile sowie den *Text der Zeile*.

Sie macht folgendes:

- Sie legt ein *neues Listen-Element* dynamisch an.
- Sie erzeugt eine *dynamisch angelegte Kopie des Zeilentextes*
  und speichert diese *im Text-Member* des neuen Listen-Elementes.
- Sie trägt den *Filenamen* und die *Zeilennummer* im neuen Listen-Element ein
  und initialisiert auch dessen *Verkettungs-Member*
  (womit, wenn das neue Element das *letzte Element der Liste* werden soll?).
- Sie ermittelt die *String-Länge i des Zeilentextes*
  und hängt das neue Element *hinten an die i-te Liste* im Listen-Array an.
  Dazu sind zwei Schritte nötig:
  - Wenn die Liste bisher *leer* ist,
    muss der *Listenkopf* auf das neue Element zeigen.

    Sonst muss die *Verkettung im bisher letzten Listen-Element*
    auf das neue Element zeigen.
  - Dann müssen wir den *Tail-Pointer* der Liste *auf das neue Element* setzen.

  Du sollst die Liste zum Anhängen *nicht* mit einer Schleife *durchlaufen*!

- Unsere nächste Funktion verarbeitet *einen File*.
  Sie wird mit unserem *Listen-Array* und dem *Filenamen* aufgerufen
  und hat *keinen Returnwert*.

  Die Funktion *öffnet* den angegebenen File zum Lesen
  (*schließe* ihn am Ende der Funktion wieder!),
  liest ihn *zeilenweise* bis zum Ende (Zeilen *mitzählen*!),
  und ruft *für jede Zeile unsere Speicher-Funktion* auf.

  Prüfe vor dem Aufruf der Speicher-Funktion, ob die Zeile *komplett gelesen* wurde
  bzw. *nicht länger* als unsere maximale Zeilenlänge ist!

- Dann schreiben wir eine *Funktion für die Ausgabe*.
  Sie wird mit unserem *Listen-Array* aufgerufen und hat *keinen Returnwert*.

  Die Funktion besteht im wesentlichen aus *zwei geschachtelten Schleifen*:
  Die *äußere* läuft über *alle Listen* unseres Listen-Arrays,
  und die *innere* geht *alle Elemente der jeweiligen Liste* durch.

  Für jedes Listen-Element wird der *Filename*, die *Zeilennummer* und der *Zeilentext ausgegeben*.

- Jetzt fehlt nur mehr das *Hauptprogramm*:
  - Es *deklariert* unser *Array von Listen-Strukturen*
    (seine Größe entspricht der maximalen Zeilenlänge)
    und initialisiert *alle Listen auf „leer“*
    (was muss man dafür wo speichern?).
  - Es prüft, ob das Programm *mit Filenamen aufgerufen* wurde,
    macht eine *Schleife über die Befehlszeile*,
    und ruft für jeden Filenamen unsere *File-Lese-Funktion* auf.
  - Dann ruft es unsere *Ausgabe-Funktion* auf.

Allgemeine Hinweise:

- ***Prüfe alle File-Operationen und Speicher-Allokationen*** auf Fehler.
  Gib im Fehlerfall *schöne Fehlermeldungen* aus (mit **`errno`**-Text usw.)
  und *beende* das Programm.

  Für die *Fehlermeldungen* ist eine *globale Variable mit dem Programmnamen* hilfreich (vergiss nicht, sie im Hauptprogramm zu initialisieren!).