Programmieren C: Strukturen/verkettete Listen, `malloc`, File-I/O: Dateien seitenweise in umgekehrter Reihenfolge 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 *ausgibt*, und zwar
  - unterteilt in *Seiten vorgegebener Zeilenanzahl*
    (ebenfalls beim Aufruf auf der Befehlszeile angegeben, vor den Filenamen)
  - in *umgekehrter Seiten-Reihenfolge* (*letzte Seite* der letzten Datei *zuerst*)
  - und mit *schönen Seitenköpfen* (Filename und Seitennummer)

  Dabei soll *jeder File auf einer neuen Seite* beginnen
  und die *letzte Seite* jedes Files *mit Leerzeilen aufgefüllt* werden,
  bis sie die vorgegebene Zeilenanzahl hat.

Nützlich ist so ein Programm z.B. zur *Ausgabe auf Druckern*,
die die Blätter mit der *bedruckten Seite nach oben* stapeln,
damit man den Stapel nach dem Drucken *nicht umsortieren* muss.

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

Für die *maximale Zeilenlänge* und für die *maximale Anzahl von Zeilen pro Seite*
darf es hingegen ein *hartcodiertes Limit* geben, die *Zeilen einer Seite* dürfen also
in einem *Array fixer Größe* gespeichert werden.

Grundidee:

- Wir verwenden eine *einfach verkettete Liste von Seiten*,
  jede Seite ist ein Listen-Element.
- Neue Seiten werden *vorne an die Liste angehängt*,
  d.h. die Liste enthält die *Seiten in umgekehrter Reihenfolge* (letzte Seite vorne).

Im Detail:

- Definiere *zwei Konstanten* für die *maximale Zeilenlänge*
  und für die *maximale Anzahl von Zeilen pro Seite*.
  Weiters brauchen wir eine *globale Variable* für die *gewünschte Seitenlänge*,
  die beim Aufruf des Programms angegeben wurde.
- Deklariere einen *Strukturtyp für ein Listenelement*, d.h. für *eine Seite*.
  Er enthält einen Pointer auf den *Filenamen*, die *Seitennummer*,
  ein *Array* von Pointern auf den *Texte der Zeilen dieser Seite*
  (dimensioniert mit der *maximalen Zeilenanzahl* pro Seite),
  und einen Pointer für die *Listen-Verkettung*.

- Als nächstes schreiben wir *zwei Hilfsfunktionen* für das Anlegen von Speicher:
  - Die eine ist nur eine *fehlergeprüfte Version* von **`strdup`**:
    Sie bekommt einen *String* übergeben und erzeugt mit **`strdup`**
    eine *dynamische Kopie* dieses Strings. Schlägt das **`strdup`** fehl,
    beendet sie das Programm mit einer *Fehlermeldung*,
    sonst liefert sie das **`strdup`**-*Ergebnis als Returnwert*.
  - Die andere erzeugt ein neues Listenelement. Sie hat *3 Parameter*:
    Den *bisherigen Head* der Seitenliste, den *Filenamen* und die *Seitennummer*.

    Die Funktion legt dynamisch ein *neues Listenelement* an, *prüft* das **`malloc`**
    auf Erfolg, und *befüllt* das neue Element *mit den drei Parametern*
    (wohin speicherst du den bisherigen Listen-Head, wenn das neue Element
    das *erste* Element der Liste werden soll?). Das *Zeilen-Array* bleibt *uninitialisiert*.

    Als *Returnwert* gibt die Funktion den *Pointer auf das neue Element* zurück
    (er ist zugleich *der neue Head* der Seitenliste).
- Unsere nächste Funktion verarbeitet *einen File*.
  Sie wird mit dem *bisherigen Head* der Seitenliste und dem *Filenamen* aufgerufen
  und liefert den *neuen Head* der Seitenliste als *Returnwert*.

  Die Funktion *öffnet* den angegebenen File zum Lesen
  (*schließe* ihn am Ende der Funktion wieder!), liest ihn *zeilenweise* bis zum Ende
  und *zählt* dabei die *Zeilen* und die *Seiten* mit.

  In der Zeilen-Lese-Schleife pasieren drei Dinge:
  - Wenn das die *allererste Zeile* dieses Files ist, oder wenn die *vorige Seite voll* ist,
    wird mit der Hilfsfunktion eine *neue Seite* vorne an die Seiten-Liste gehängt
    (merk dir den *Returnwert* der Hilfsfunktion als *neuen Listen-Head*!),
    die *Seitennummer erhöht*, und eventuell die Zeilennummer wieder auf 0 gesetzt.
  - Es wird geprüft, ob die neue *Zeile vollständig gelesen* wurde oder *zu lang* war.
  - Mit unserer **`strdup`**-Hilfsfunktion wird eine *dynamische Kopie der Zeile* erzeugt
    und im *nächsten freien Element des Zeilen-Arrays* der aktuellen Seite gespeichert.

  Es ist dir überlassen, ob du die *Zeilennummern* von File-Anfang bis Ende zählst und
  aus der File-bezogenen Zeilennummer die Nummer der Zeile innerhalb der Seite
  berechnest (wie?), oder ob du nur die Zeilennummer innerhalb der Seite zählst
  und bei jeder neuen Seite wieder bei 0 anfängst.

  Nach der Zeilen-Lese-Schleife muss mit einer weiteren Schleife eine *Leerzeile*
  in *alle noch nicht belegten Zeilen* der aktuellen Seite gespeichert werden.
- Dann schreiben wir eine *Funktion für die Ausgabe*.
  Sie wird mit unserem *Listen-Head* aufgerufen und hat *keinen Returnwert*.

  Die Funktion besteht im wesentlichen aus *zwei geschachtelten Schleifen*:
  Die *äußere* läuft über *alle Seiten* (d.h. alle Elemente unserer Liste),
  und die *innere* geht *das Zeilen-Array in jeder Seite durch* und *gibt alle Zeilen aus.*

  Vergiss nicht, *vor* der inneren Schleife einen *schönen Seitenkopf* auszugeben!

- Jetzt fehlt nur mehr das *Hauptprogramm*:
  - Es deklariert den *Head unserer Liste* und setzt die Liste auf *leer* (wie?).
  - Es prüft, ob das Programm *mit der gewünschten Seitenlänge* sowie *mindestens einem Filenamen* aufgerufen wurde, *speichert* die Seitenlänge in der globalen Variablen, und *prüft*, ob sie größer 0, aber nicht größer als die maximale Seitenlänge ist.
  - Dann macht es eine *Schleife über die restlichen Worte der Befehlszeile* und ruft für jeden Filenamen unsere *File-Lese-Funktion* auf (und merkt sich jedesmal den *Returnwert als neuen Head*).
  - 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!).