Programmieren C: Strukturen, Pointer, `malloc`: Verkettete Liste, sortiert

Klaus Kusche

Gesucht ist ein Programm, das mit *beliebig vielen Zahlen auf der Befehlszeile* aufgerufen wird und eine Liste dieser Zahlen ausgibt:

- Die Liste soll *aufsteigend sortiert* sein.
- Die Liste soll *keine doppelten Werte* enthalten: Jeder Wert wird *nur einmal* ausgegeben, und nach jedem Wert wird ausgegeben, *wie oft* er vorgekommen ist.
- Die Liste soll nicht nur einmal nach dem Einlesen aller Zahlen ausgegeben werden, sondern zur Kontrolle jedesmal, nachdem eine Zahl hinzugefügt worden ist.

Wir wollen diesmal aber *kein Array und keine Sortierfunktion* verwenden: Es wäre mühsam und aufwändig, nach jeder neuen Zahl das Array frisch zu sortieren oder die Zahl an der richtigen Stelle einzufügen und den Rest des Arrays nach hinten zu schieben.

Stattdessen verwenden wir eine ***einfach verkettete, sortierte Liste***:

- Jedes Element der Liste wird *einzeln* mit **`malloc`** dynamisch angelegt.
- Eine Variable enthält einen *Pointer auf das erste (hier: kleinste) Element der Liste*. Dieser Pointer heißt üblicherweise **`head`** („Kopf“ der Liste).

  Enthält **`head`** den **`NULL`**-Pointer, so ist die Liste *leer* (enthält keine Elemente).
- Jedes Element der Liste ist eine *Struktur* und enthält als Member u.a. einen *Pointer auf das nächste Element* der Liste. Dieser Pointer heißt meist **`next`**.
- Im *letzten Element* der Liste enthält **`next`** den **`NULL`**-Pointer (==› *„kein Nachfolger“*).

Ausgehend von **`head`** kann man der Reihe nach alle Elemente der Liste *durchlaufen*, indem man bei dem Element beginnt, auf das **`head`** zeigt, und einfach jedesmal dem **`next`**-Pointer des aktuellen Elementes zum nächsten Element folgt, und zwar so lange, bis der aktuelle Pointer **`NULL`** ist.

In einer solchen Liste läßt sich auch relativ leicht an beliebiger Stelle *einfügen*, *ohne* bestehende Element zu *verschieben*: Man muss nur die Pointer-Verkettung „umbiegen“.

Wir brauchen für das Programm also einen *Struktur-Typ für die einzelnen Listen-Elemente*, der Folgendes enthält:

- Die *Zahl*, die in diesem Element gespeichert ist (**`int`**).
- Einen *Zähler*, *wie oft* diese Zahl bisher vorgekommen ist.
- Einen *Zeiger* auf das nächste Element (das *wieder eine solche Struktur* ist).

Wir brauchen 3 Funktionen:

- **neu:** Diese Funktion soll einen *Pointer auf ein neu im Speicher angelegtes Element* als *Returnwert* zurückliefern. Sie soll das neue Element auch gleich initialisieren.

  Dafür bekommt sie *zwei Parameter*: Die *Zahl* und den *Nachfolge-Pointer*, die im neuen Element gespeichert werden sollen.
  Den *Zähler* im neuen Element soll sie immer auf **`0`** setzen.

Wenn das **malloc** keinen freien Speicherplatz mehr findet,
soll die Funktion das Programm mit einer *Fehlermeldung* beenden.

- **find**: Diese Funktion wird mit *einer Liste* (= dem **head**-Pointer der Liste)
  und *einer Zahl* als *Parameter* aufgerufen und soll diese Zahl in der Liste *suchen*:
  - Ist die Zahl schon in der Liste *enthalten*, soll ein
    Pointer auf das bestehende Element als Returnwert zurückgeliefert werden.
  - Kommt sie *noch nicht* vor, soll ein *neues Element* (mit der Funktion **neu**)
    angelegt und an der richtigen Stelle in der Liste *eingefügt* werden und dann
    ein *Pointer auf das neue Element als Returnwert* zurückgegeben werden.

  Da die Funktion ja eventuell ein *neues vorderstes Element* in die Liste einfügt,
  muss sie den **head**-Pointer *im Aufrufer ändern* können.
  Dieser muss daher *„by Reference"* übergeben werden.
  Wie sieht daher die Parameter-Deklaration aus?
  Wie muss in der Funktion auf den **head**-Parameter zugegriffen werden?

  Die Funktion **find** muss *mehrere Fälle* unterscheiden:

  - Die *Liste ist leer* oder die *Zahl im ersten Element ist schon größer*
    als die gesuchte Zahl:

    In beiden Fällen muss ein *neues erstes Element* mit der neuen Zahl
    ganz vorne (d.h. als neuer **head**) eingefügt werden.
    Der *Nachfolger* des neuen Elementes ist der bisherige **head**
    (egal ob dieser **NULL** war oder nicht).

  - Sonst *durchläuft* man die Liste Element für Element.

    Für das Ende dieser Schleife gibt es *zwei* Möglichkeiten:

    - Die Zahl im aktuellen Element ist *gleich* der gesuchten Zahl:

      In diesem Fall wird der *Pointer auf das aktuelle Element zurückgegeben*.

    - Man ist beim *letzten Element* der Liste (d.h. das aktuelle Element
      hat *keinen Nachfolger*), oder die *Zahl im Nachfolger*
      des aktuellen Elementes ist *größer als die gesuchte Zahl*:

      In beiden Fällen muss ein *neues Element* mit der gesuchten Zahl
      hinter dem aktuellen Element eingefügt werden:

      - Der *Nachfolger* des *neuen* Elementes ist der *bisherige Nachfolger*
        des aktuellen Elementes (das kann auch **NULL** sein,
        wenn man ein neues letztes Element anhängt).

      - Der neue *Nachfolger* des *aktuellen* Elementes
        wird das *neu angelegte Element*.

      Als *Returnwert* wird der Pointer auf das *neue* Element zurückgegeben.

- **print** (mit **head** als Parameter, ohne Returnwert):

  Diese Funktion *durchläuft* die Liste einmal vom Anfang bis zum Ende
  und *gibt jede Zahl samt ihrem Zähler aus*.

Unser **Hauptprogramm** enthält den **`head`**-Pointer unserer Liste.
Wie *initialisierst* du ihn, wenn die Liste beim Start des Programmes *leer* sein soll?

Dann macht es in einer *Schleife* für *jede Zahl auf der Befehlszeile* Folgendes:

- Die Zahl *einlesen* und mittels **`find`**-Funktion in der Liste *suchen bzw. einfügen*.
- Den *Zähler des gefundenen Elementes* (auf das der Returnwert von **`find`** zeigt) um eins *erhöhen*.
- Die Liste mittels **`print`**-Funktion *ausgeben*.