AIK Programmieren 1 Übung: Verkettete Listen

Klaus Kusche

Anmerkung: Das ist Studenten-Stoff, *freiwillige* Übung nur für die ganz Guten!

Sortierte Liste von Zahlen

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* im Speicher angelegt (Details siehe unten!).
- Eine *globale* Variable[1] enthält einen *Pointer auf das erste (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.
- Einen *Zähler*, wie oft sie bisher vorgekommen ist.
- Einen *Zeiger* auf das nächste Element (das *wieder eine solche Struktur* ist).

---

1 Stilistisch wäre es besser, wenn man **head** als *lokale Variable* im **main** deklariert und den Funktionen als *Parameter* übergibt. Da die Einfüge-Funktion **head** aber ändern muss, müsste **head** *„by Reference“*, also als *Pointer auf einen Pointer*, übergeben werden. Diese Verkomplizierung sparen wir uns vorläufig.

Weiters müssen wir **head** ***global*** als Pointer auf eine solche Struktur deklarieren und auf **NULL** initialisieren, da die Liste beim Start des Programmes ja leer sein soll.

Wir brauchen 4 Funktionen:

- **neu**: Diese Funktion soll einen *Pointer auf ein neu im Speicher angelegtes Element* als *Returnwert* zurückliefern.

  Folgendes Konstrukt reserviert genau so viel neuen Speicherplatz, wie für eine Struktur (ein Element) notwendig ist, und speichert einen Pointer auf diesen neu angelegten Platz in **p**:

  `p = (struct eintrag *) (malloc(sizeof(struct eintrag)));`
  (wobei du den Namen des **`struct`**-Typs anpassen musst!)

  Wird dabei **NULL** in **p** gespeichert, so hat das **`malloc`** keinen freien Speicher mehr gefunden. Prüfe das, das Programm soll in diesem Fall mit einer *Fehlermeldung* enden.

  Der neu angelegte Speicher ist uninitialisiert (enthält zufällige Werte), unsere Funktion soll im neuen Element daher auch gleich *die richtigen Werte speichern*.

  Die Funktion bekommt dafür *zwei Parameter*: Die *Zahl* und den *Nachfolge-Pointer*, die im neuen Element gespeichert werden sollen.
  Der Zähler im neuen Element wird immer auf 0 gesetzt.

- **`find`**: Diese Funktion wird mit einer *Zahl als Prameter* aufgerufen und soll diese Zahl in der Liste *suchen*:
  - Kommt die Zahl schon vor, 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.

  Diese Funktion muss mehrere Fälle unterscheiden:
  - Die *Liste ist leer oder die Zahl im ersten Element ist schon größer* als die gesuchte Zahl:

    In diesem Fall muss ein *neues erstes Element ganz vorne* (d.h. als neuer **head**) mit der neuen Zahl eingefügt werden. Der *Nachfolger* des neuen Elementes ist der bisherige **head** (egal ob er **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 (überlege, woran man das erkennt, denn wenn der aktuelle Pointer **NULL** ist, bist du schon ein Element zu weit gelaufen!), oder die Zahl im *Nachfolger* des aktuellen Elementes ist *größer als die gesuchte Zahl* (überlege: Wie kommst du von einem Pointer auf das aktuelle Element auf die Zahl in dessen Nachfolger?):

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, und ein Pointer auf das neue Element wird als Returnwert zurückgegeben.

- **`print`** (ohne Argumente und ohne Returnwert):
  Diese Funktion *durchläuft* die Liste einmal vom Anfang bis zum Ende (wie oben beschrieben) und gibt jede Zahl samt ihrem Zähler aus.
- **`main`**: **`main`** macht 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 1 *erhöhen*.
  - Die Liste mittels **`print`**-Funktion *ausgeben*.