Dynamische Datenstrukturen

1 Das Problem mit statischen Datenstrukturen

Wir haben gesehen, dass bei einem Array der Platz fix vorgegeben werden muss. Beim Array handelt es sich auch um eine primitive Datenstruktur. Einen fixen Speicherplatz zu belegen ist vor allem dann ein Problem, wenn wir es mit dynamisch wachsenden und schrumpfenden Datenmengen zu tun haben.

In der Informatik verwendet man deshalb Listen und Bäume, um eine flexible Datenstruktur zu haben.

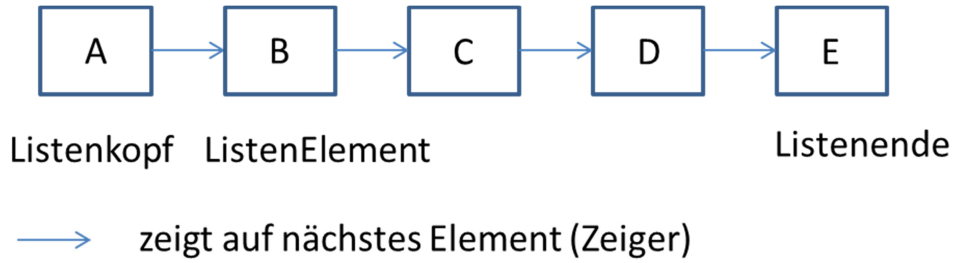
2 Listen für dynamische Datenstrukturen

Listen sind sehr wichtig und werden z.T. bereits als Grundelement der Sprache eingebaut. Programmiersprachen, die Listen oder dynamische Datenstrukturen als Grundelement anbieten, lassen dem Programmierer keinen direkten Zugriff auf die Speicherplatzverwaltung (also Speicherplatz reservieren und wieder freigeben).

Das mag dann etwas eleganter sein fürs Programmieren, ist aber weniger effizient. Deshalb hat das C nicht, weil Effizienz bei C sehr wichtig ist. Somit müssen solche Strukturen selber programmiert werden (es gibt aber genügend Beispiele !)

3 Wie sieht eine solche Liste aus ?

Das Prinzip einer Liste ist wie eine Kette (deshalb eine „verkettete Liste“):



Wir haben 3 Elemente:

Einen **Listenkopf**
Ein (oder mehrere) **Listenelement**(e)
Und einen **Zeiger**

Das letzte Element (Listenende) zeigt einfach auf NULL und ist deshalb wiederum ein Listenelement.

4 Wie suchen wir in der Liste?

Beim Array können wir direkt mittels Index auf ein bestimmtes Element zugreifen:

```
int myNumber = number[7];
```

Das geht leider bei der verketteten Liste nicht. Hier müssen wir die Liste durchgehen und jedes einzelne Element „anschauen“, bis wir das gesuchte Element gefunden haben. Dafür ist das Einfügen und Löschen in einer Liste je nach Fall einfacher, weil wir dann bloss ein Element hinzufügen (ans Ende) oder beim Entfernen eines Elements auf das nächste zeigen.

5 Aufbau einer verketteten Liste in C

Prinzip der verketteten Liste: jede Struktur beinhaltet einen Verweis auf das nächste Element (welches wiederum eine Struktur ist).

Somit werden die Listenelemente als **Strukturen** in C definiert (deshalb sind Strukturen so wichtig !):

Ebenfalls benützt die verkettete Liste Pointer (**Zeiger**), um die Elemente zu verknüpfen.

Beispiel:

```
struct angestellt {
   char name[40];
   char vorname[40];
   struct datum alter;
   struct datum eingest;
   long gehalt;
   struct angestellt *next;   //verweis auf das nächste element
};
```

Die letzte Zeile in dieser Struktur ist sehr wichtig, weil sie das Prinzip der verketteten Liste umsetzt: sie zeigt auf das nächste Element, welches wiederum eine Struktur „angestellt“ ist.

Wie arbeitet man mit Strukturen?

1. **Anfangswert auf NULL setzen:**

   Beide Strukturen werden zuerst auf NULL gesetzt:

   ```
   struct angestellt *next = NULL;
   struct angestellt *anfang = NULL;
   ```

2. **Speicherplatz für Struktur reservieren mittels Funktion „malloc“:**

   ```
   anfang = malloc(sizeof(struct Element));
   anfang->next = NULL;
   ```

Hinweis zu **Operator „->“.** Wenn man mit Zeiger auf Strukturen arbeitet, dann vereinfacht dieser Operator den Zugriff. Man kann den Operator auch als „zeige auf“ lesen.

3. **Eingabe in Elemente hinzufügen**

Prinzip: Prüfen, ob das erste Element noch NULL ist oder nicht. Wenn es NULL ist, dann werden im ersten Element die Daten abgefüllt.
Sonst gehen wir auf das nächste Element und reservieren für dieses einen Speicherplatz. Dann wird dieses Element abgefüllt.

4. Löschen des ersten Elements

Prinzip: Wenn das erste Element gelöscht wird, dann ist das nächste Element nun das erste. Das muss aber so programmiert werden!

Speicherplatz freigeben vom ersten Element (= erstes Element wird physisch gelöscht):

```
free(anfang);
```

Und das zweite Element ist jetzt der Anfang, die Adresse des Zeigers vom zweiten Element wird an den Anfang übergeben.

5. Löschen eines Elements irgendwo in der Liste

Das ist schwieriger, weil wir auf keinem Fall den Speicherplatz des gelöschten Elements einfach so freigeben dürfen, sonst ist die Kette unterbrochen.
Wir müssen das aktuelle Element mit dem Zeiger des zu löschenden Elements verknüpfen:

```
element->next = element1 -> next;
```

Element1 wird gelöscht und bevor wir das tun, wird der Zeiger auf das nächste Element dem aktuellen Zeiger von „element“ zugewiesen. Somit bleibt die Kette intakt und wir können den Speicherplatz von element1 freigeben.

Beispiele um das zu üben:

Ein Beispiel für verkettete Liste finden Sie im Modul 118 Script, in Kapitel 8. Ebenfalls ist dort nochmals eine Repetition zum Thema Zeiger.

Eine sehr gute Einführung findet sich hier:
http://openbook.galileocomputing.de/c_von_a_bis_z/021_c_dyn_datenstrukturen_001.htm#mja75ba2f4ab8f95e9e321d195c1e26d76

Hier wird auch Schritt für Schritt das Funktionieren einer Liste erklärt.

Ebenfalls hat auch Niklas Liechti eine sehr gute Umsetzung gemacht. Das Beispiel kann ich Ihnen (bzw. Niklas Liechti) zeigen.

6 Aufbau einer verketteten Liste in C++ / Java

Für alle, die mit C++ oder Java arbeiten: versuchen Sie herauszufinden, was es für Möglichkeiten gibt.
Bei Java gibt es sogenannte Klassenbibliotheken, welche mehrere Variante anbieten (Stichwort: „Collections“).