

Hardwarenahe Programmierung / Angewandte Informatik

Übungsaufgaben – 22. Januar 2018

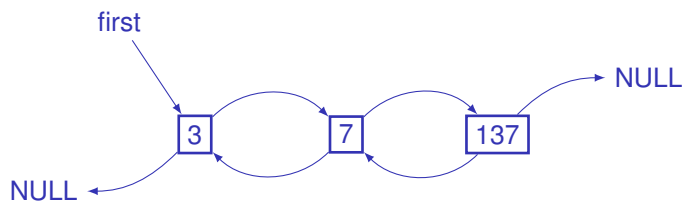
Diese Übung enthält Punkteangaben wie in einer Klausur. Um zu „bestehen“, müssen Sie innerhalb von 60 Minuten unter Verwendung ausschließlich zugelassener Hilfsmittel 10 Punkte (von insgesamt 21) erreichen.

Aufgabe 1: Doppelt verkettete Liste

In der Vorlesung wurde ein Beispiel-Programm ([lists-11.c](#)) zur Verwaltung einfach verketteter Listen erstellt (an die Liste anhängen, in die Liste einfügen, Liste ausgeben usw.).

- (a) Ergänzen Sie eine Funktion `delete_from_list()` zum Löschen eines Elements aus der Liste mit Freigabe des Speicherplatzes. (3 Punkte)
- (b) Ergänzen Sie eine Funktion `print_reverse()` die die Liste in umgekehrter Reihenfolge ausgibt. (Hinweis: Rekursion) (3 Punkte)

Eine doppelt verkettete Liste hat in jedem Knotenpunkt (`node`) *zwei* Zeiger – einen auf das nächste Element (`next`) und einen auf das vorherige Element (z. B. `prev` für „previous“). Dadurch ist es leichter als bei einer einfach verketteten Liste, die Liste in umgekehrter Reihenfolge durchzugehen.



Der Rückwärts-Zeiger (`prev`) des ersten Elements zeigt, genau wie der Vorwärts-Zeiger (`next`) des letzten Elements, auf *nichts*, hat also den Wert `NULL`.

- (c) Schreiben Sie das Programm um für doppelt verkettete Listen. (5 Punkte)

Aufgabe 2: Ternärer Baum

Der in der Vorlesung vorgestellte *binäre Baum* ist nur ein Spezialfall; im allgemeinen können Bäume auch mehr als zwei Verzweigungen pro Knotenpunkt haben. Dies ist nützlich bei der Konstruktion *balancierter Bäume*, also solcher, die auch im *Worst Case* nicht zu einer linearen Liste entarten, sondern stets eine – möglichst flache – Baumstruktur behalten.

Wir betrachten einen Baum mit bis zu drei Verzweigungen pro Knotenpunkt, einen sog. *ternären Baum*. Jeder Knoten enthält dann nicht nur einen, sondern *zwei* Werte als Inhalt:

```
typedef struct node
{
    int content_left, content_right;
    struct node *left, *middle, *right;
} node;
```

Wir konstruieren nun einen Baum nach folgenden Regeln:

- Innerhalb eines Knotens sind die Werte sortiert: `content_left` muß stets kleiner sein als `content_right`.
- Der Zeiger `left` zeigt auf Knoten, deren enthaltene Werte durchweg kleiner sind als `content_left`.
- Der Zeiger `right` zeigt auf Knoten, deren enthaltene Werte durchweg größer sind als `content_right`.
- Der Zeiger `middle` zeigt auf Knoten, deren enthaltene Werte durchweg größer sind als `content_left`, aber kleiner als `content_right`.
- Ein Knoten muß nicht immer mit zwei Werten voll besetzt sein; er darf auch *nur einen* gültigen Wert enthalten.

Der Einfachheit halber lassen wir in diesem Beispiel nur positive Zahlen als Werte zu. Wenn ein Knoten nur einen Wert enthält, setzen wir `content_right = -1`, und der Zeiger `middle` wird nicht verwendet.

- Wenn wir neue Werte in den Baum einfügen, werden *zuerst* die nicht voll besetzten Knoten aufgefüllt und *danach erst* neue Knoten angelegt und Zeiger gesetzt.
- Beim Auffüllen eines Knotens darf nötigenfalls `content_left` nach `content_right` verschoben werden. Ansonsten werden einmal angelegte Knoten nicht mehr verändert.

(In der Praxis dürfen Knoten gemäß speziellen Regeln nachträglich verändert werden, um Entartungen gar nicht erst entstehen zu lassen – siehe z. B. <https://de.wikipedia.org/wiki/2-3-4-Baum>.)

- (a) Zeichnen Sie ein Schaubild, das veranschaulicht, wie die Zahlen 7, 137, 3, 5, 6, 42, 1, 2 und 12 nacheinander und in dieser Reihenfolge in den oben beschriebenen Baum eingefügt werden – analog zu den Vortragsfolien ([hp-20180129.pdf](#)), Seite 20. (3 Punkte)
- (b) Dasselbe, aber in der Reihenfolge 2, 7, 42, 12, 1, 137, 5, 6, 3. (3 Punkte)
- (c) Beschreiben Sie in Worten und/oder als C-Quelltext-Fragment, wie eine Funktion aussehen müßte, um den auf diese Weise entstandenen Baum sortiert auszugeben. (4 Punkte)

Viel Erfolg – auch in der Klausur!