PR3 - Leseaufgabe zur Vorlesung 8

(Stand 2021-10-01 14:51)

Prof. Dr. Holger Peine
Hochschule Hannover
Fakultät IV – Abteilung Informatik
Raum 1H.2.60, Tel. 0511-9296-1880
Holger.Peine@hs-hannover.de

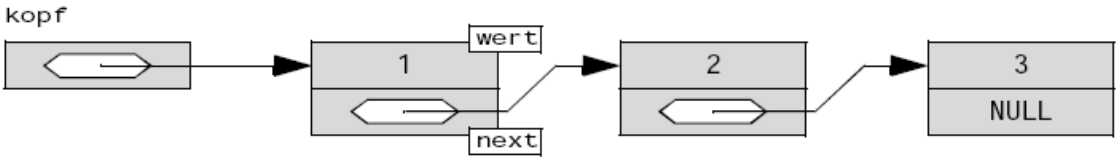
L.8 Dynamische Datenstrukturen

L.8.1 Einfach verkettete Liste

In der Vorlesung haben wir die einfach verkettete Liste wie folgt realisiert:

```
struct knoten {
    int wert; /* oder komplexere Daten */
    struct knoten* next;
};

...
int main(void) {
  struct knoten* kopf = ...;
  ...
}
```



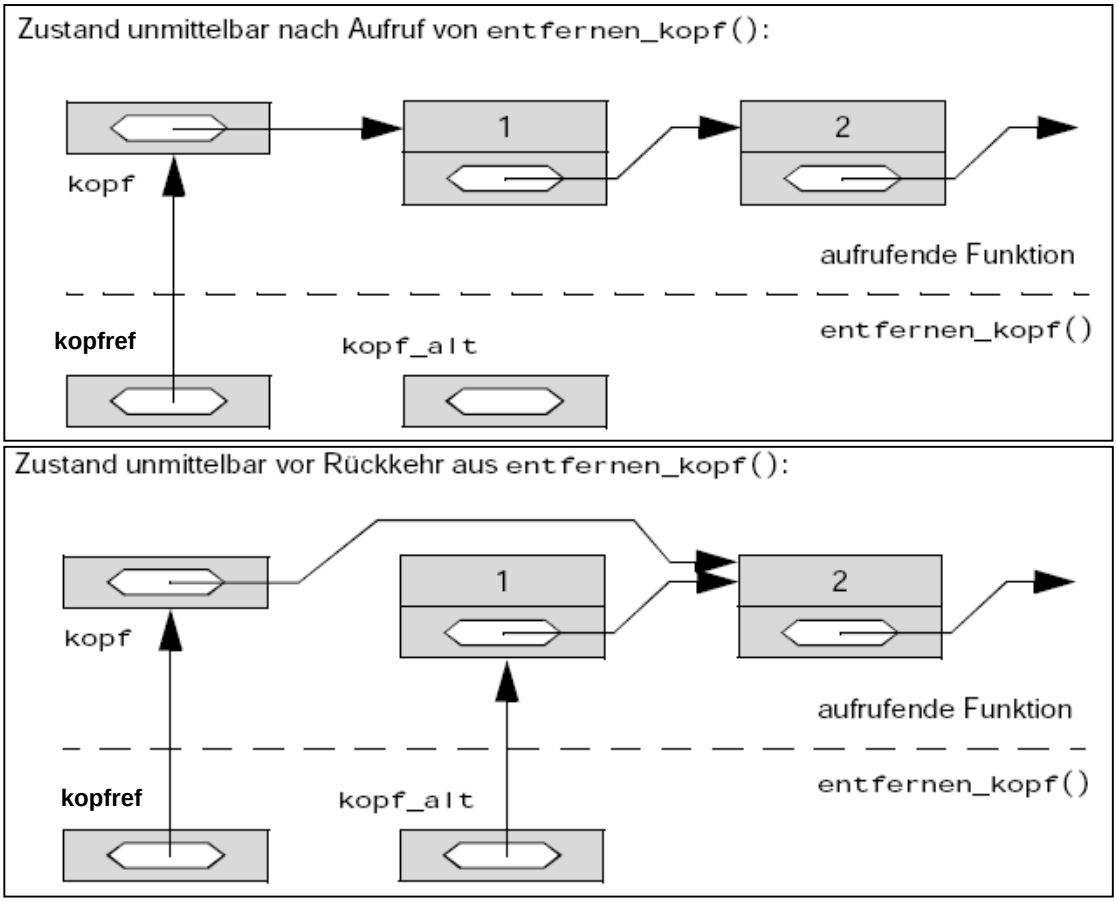
Wir haben Funktionen zum Durchlaufen und Suchen sowie zum Einfügen von Knoten vorgestellt. Hier folgen nun weitere Funktionen zum Entfernen von Listenknoten.

L.8.1.1 Entfernen des Listenkopfes

Wenn der Listenkopf entfernt werden soll, muss der Parameter der Funktion ein call-by-reference-Parameter sein, da der Kopfzeiger verändert werden muss:

```
/* Liefert den entfernten Knoten oder NULL */
struct knoten* entfernen_kopf(struct knoten** kopfref) {
    struct knoten* kopf_alt;
    if ((kopfref == NULL)||(*kopfref == NULL)) return NULL;
    kopf_alt = *kopfref;
    *kopfref = (*kopfref)->next;
    return kopf_alt;
}
```

In der Variable `kopf_alt` merken wir uns den alten Kopf-Zeiger, um ihn zurückgeben zu können. Der Kopfzeiger soll nach Rückkehr auf den zweiten Knoten zeigen (`(*kopfref)->next`).

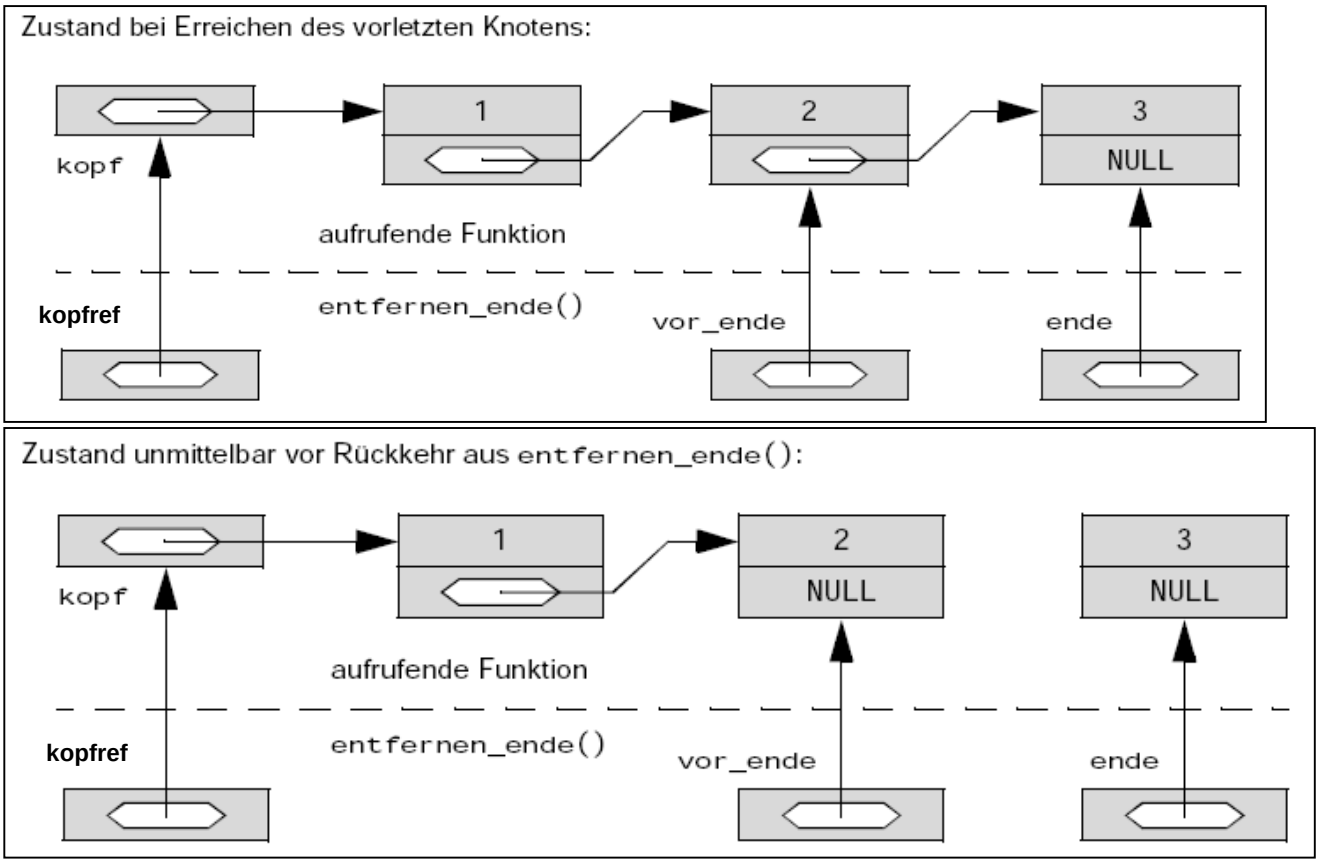


L.8.1.2 Entfernen des Knotens am Listenende

Um den letzten Knoten zu entfernen, müssen wir vorher durch die Liste durchwandern, bis wir das Ende gefunden haben. Wichtig: Wir müssen am vorletzten Knoten halt machen, denn genau hier müssen wir Zeiger „verbiegen“.

```
/* Liefert den entfernten Knoten oder NULL */
struct knoten* entfernen_ende(struct knoten** kopfref) {
    struct knoten *vor_ende, *ende;
    if ((kopfref==NULL)||(*kopfref==NULL)) return NULL; /*Fehler oder leere Liste*/
    if ((*kopfref)->next==NULL) { /* Einelementige Liste */
        ende = *kopfref;
        *kopfref = NULL;
        return ende;
    }
    /* Liste hat mindestens zwei Elemente */
    vor_ende = *kopfref;  /* vor_ende soll auf den vorletzten Knoten zeigen */
    while (vor_ende->next->next != NULL) {
        vor_ende = vor_ende->next;
    }
    ende = vor_ende->next;
    vor_ende->next = NULL;
    return ende;
}
```

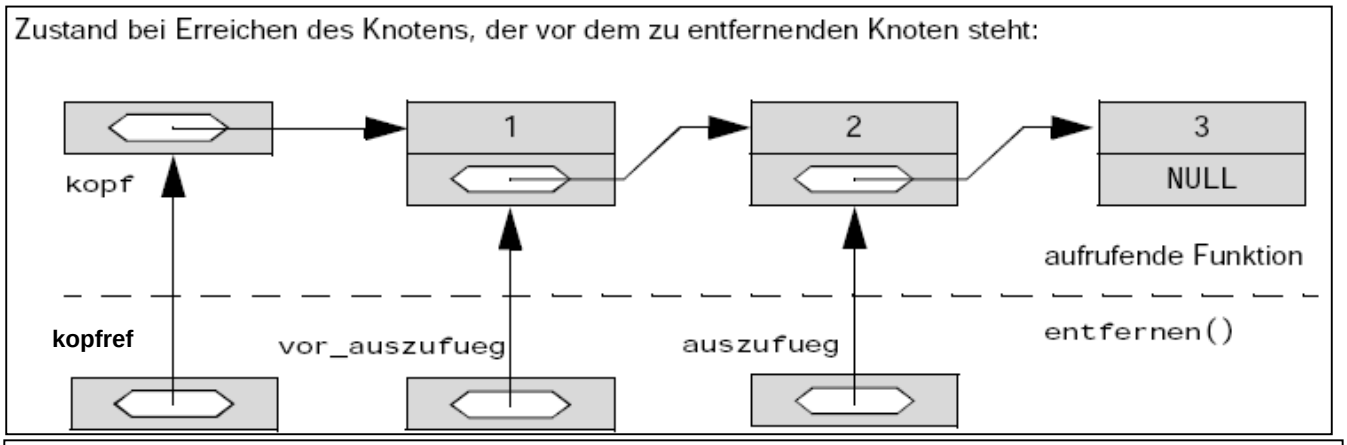
Sonderfälle betreffen die leere Liste (hier ist nichts zu tun) und die einelementige Liste (hier muss lediglich der Kopfzeiger auf `NULL` gesetzt werden).

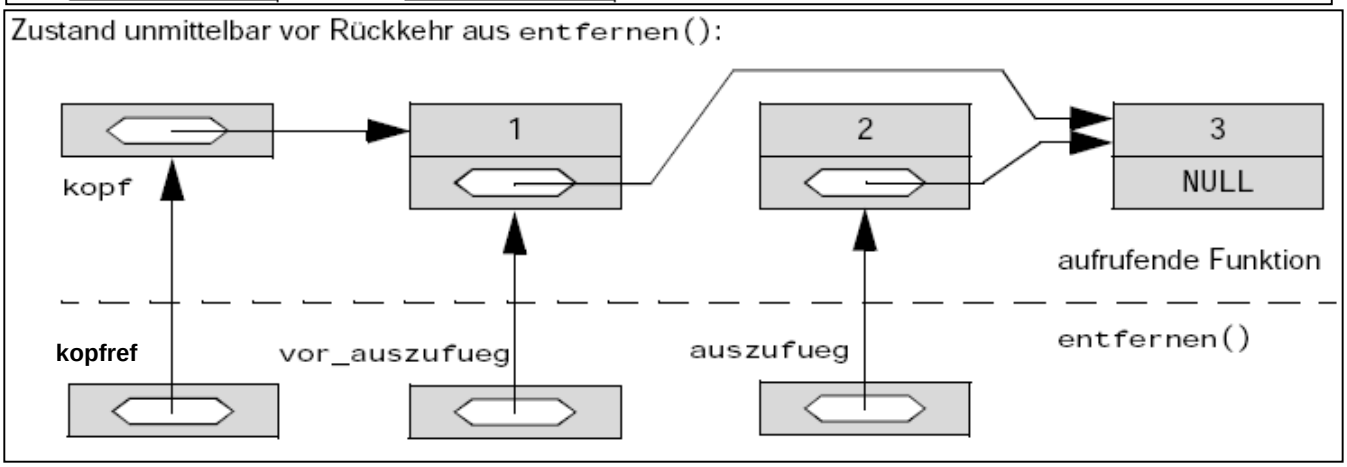


L.8.1.3 Entfernen eines bestimmten Knotens

Hier müssen wir ähnlich vorgehen, wie beim Entfernen am Ende. Unterschied: Wir suchen nicht den vorletzten Knoten, sondern den Vorgänger des zu entfernenden Knotens.

```
/* Liefert den entfernten Knoten oder NULL */
struct knoten* entfernen(struct knoten** kopfref, struct knoten* auszufueg) {
    struct knoten* vor_auszufueg;
    if ((auszufueg==NULL)||(kopfref==NULL)||(*kopfref==NULL)) return NULL;
    if (auszufueg==*kopf) {
        *kopfref = (*kopfref)->next;
        return auszufueg;
    }
    vor_auszufueg = *kopfref;
    while (vor_auszufueg->next!=auszufueg) {
        if (vor_auszufueg->next == NULL) return NULL; /* nicht gefunden */
        vor_auszufueg = vor_auszufueg->next;
    }
    vor_auszufueg->next = auszufueg->next;
    return auszufueg;
}
```





L.8.1.4 Was passiert mit dem entfernten Knoten?

Die drei vorgestellten Funktionen entfernen einen Knoten (genauer: sein `struct`-Objekt) zwar aus der Liste, löschen die Struktur aber nicht aus dem Speicher. Die aufrufende Funktion kann also über die Struktur weiter verfügen oder sie selbst durch `free()` löschen. Das ist auch konsistent mit der Erzeugung von Knoten: Unsere Listenfunktionen erzeugen keine Knoten, sondern erwarten und liefern Knoten. Die Erzeugung und Löschung übernimmt immer der Aufrufer.

L.8.2 Weitere Datenstrukturen

Weitere Strukturen (doppelt verkette Listen, Bäume, Hashtabellen, etc.) würden den Rahmen dieser Veranstaltung sprengen. Interessierten seien die folgenden Bücher empfohlen:

- Vogt: C für Java-Programmierer (Details siehe Literaturliste)
- Sedgewick, R.: Algorithms in C. 3rd Ed. Addison-Wesley 1998.