## Einfach verkettete Listen in C

Da in der beigefügten Literatur die Programmbeispiele in Pascal geschrieben sind, haben wir hier einige dieser Funktionen in C nachgebildet. Diese Funktionen verdeutlichen nur die Grundprinzipien und sind ggf. nicht die günstigsten Implementierungen der Listenoperationen. Alle Funktionen gehen davon aus, daß kein Kopfelement (ohne echtes Datum) verwendet wird (im Unterschied zu einigen der Implementierungen in Pascal). Kopfelemente haben Vor- und Nachteile (welche?). Ob Sie ein Kopfelement verwenden wollen oder nicht, ist Ihnen überlassen.

Typdefinitionen für einfach verkettete Listen:

```
typedef struct listelem *LISTPTR;

typedef struct listelem {
  LISTPTR  next;
  DATATYPE datum;
} LISTELEM;
```

Funktion zum Testen, ob eine einfach verkettete Liste leer ist (ohne Verwendung eines leeren Kopfelementes):

```
int is_empty (LISTPTR list)
{
  return (list == NULL);
}
```

Funktion zum Testen, ob die aktuelle Position die letzte in einer einfach verketteten Liste ist (`pos` wird als ungleich `NULL` angenommen):

```
int is_last (LISTPTR pos)
{
  return (pos->next == NULL);
}
```

Funktion zum Auffinden eines gegebenen Datums in einer einfach verketteten Liste (ohne Ausnutzung der „short circuiting evaluation"):

```
LISTPTR find (LISTPTR list, DATATYPE datum)
{
  LISTPTR pos = list;

  while (pos != NULL) {
    if (pos->datum = datum) break;
    pos = pos->next;
  }
  return pos;
}
```

Funktion zum Auffinden eines gegebenen Datums in einer einfach verketteten Liste (mit Ausnutzung der „short circuiting evaluation“):

```
LISTPTR find (LISTPTR list, DATATYPE datum)
{
  LISTPTR pos = list;

  while ((pos        != NULL)
  &&     (pos->datum != datum))
    pos = pos->next;
  return pos;
}
```

Die „short circuiting evaluation“ (siehe Programm `power.c` auf der bekannten WWW-Seite) gehört in C zur Standard-Sprachdefinition. Sie ist daher in C portabel.

Funktion zum Einfügen eines Datums in eine einfach verkettete Liste:

```
LISTPTR insert (LISTPTR list, LISTPTR pos, DATATYPE datum)
{
  LISTPTR tmp;

  tmp = (LISTPTR)malloc(sizeof(LISTELEM));
  if (!tmp) return NULL;
  tmp->datum = datum;
  if (pos == NULL) {
    tmp->next = list;
    return tmp;
  }
  tmp->next = pos->next;
  pos->next = tmp;
  return list;
}
```

Wird als Position `pos` der Wert `NULL` übergeben, so wird am Kopf der Liste eingefügt, sonst hinter dem Element, auf das `pos` zeigt.

Funktion zum Löschen eines gegebenen Datums aus einer einfach verketteten Liste (ohne Verwendung eines leeren Kopfelementes):

```
LISTPTR delete (LISTPTR list, DATATYPE datum)
{
  LISTPTR pos = list;
  LISTPTR tmp;

  if (pos == NULL) return NULL;
  if (pos->datum == datum) {
    list = list->next;
    free(pos); }
  else {
```

```
    while ((pos->next != NULL)
    &&     (pos->next->datum != datum))
      pos = pos->next;
    if (pos->next != NULL) {
      tmp = pos->next;
      pos->next = tmp->next;
      free(tmp);
    }
  }
  return list;
}
```

Diese Funktion gibt den ggf. geänderten Zeiger auf das erste Listenelement zurück. Man beachte, daß dieser nicht mit dem an die Funktion übergebenen Zeiger identisch sein muß, da sich das zu löschende Datum auch im ersten Listenelement befinden kann.

Funktion zum Löschen eines gegebenen Datums aus einer einfach verketteten Liste (ohne Verwendung eines leeren Kopfelementes, unter Verwendung eines Zeigers auf einen Zeiger):

```
LISTPTR delete (LISTPTR list, DATATYPE datum)
{
  LISTPTR *p = &list;
  LISTPTR tmp;

  while ((*p         != NULL)
  &&     ((*p)->datum != datum))
    p = &(*p)->next;
  if (*p != NULL) {
    tmp = *p;
    *p  = tmp->next;
    free(tmp);
  }
  return list;
}
```

Auch diese Funktion gibt den ggf. geänderten Zeiger auf das erste Listenelement zurück.

Funktion zum Löschen einer einfach verketteten Liste:

```
void delete_list (LISTPTR list)
{
  LISTPTR tmp;

  while (list != NULL) {
    tmp  = list;
    list = list->next;
    free(tmp);
  }
}
```