

Beispiele zu einfach / doppelt verketteten Listen

```
/*
Skript S. 15 - 6

Liest Daten in eine einfach verkettete Liste ein.
Diese wird nach Menge der Daten dynamisch erweitert.

Nachteil dieser Implementierung zur besseren Verständlichkeit:
Es wird ein Element zu viel angelegt.
*/

#include <stdio.h>          // Ein-/Ausgabe
#include <malloc.h>        // Speicherverwaltung (u.a. malloc)

// Definition eines Listenelements

struct DATEN {

// Datenteil

    int KdNr;
    char Name[30+1];

// Zeiger auf nächstes Listenelement

    struct DATEN *next;
};

void main ()
{

// Zeiger auf den Listenstart und Zeiger auf das Element,
// mit dem aktuell gearbeitet wird

    struct DATEN *Start = NULL, *aktuell = NULL;

// Anlegen eines neuen Listenelements (Achtung, es hat keinen Namen,
// daher wird Referenz (Pointer) benötigt)
// Start verweist auf dieses Element

    Start = (struct DATEN * ) malloc(sizeof(DATEN));

// wir beginnen das Bearbeiten der Liste am Anfang, setzen also
// auch aktuell auf Start

    aktuell = Start;

// solange noch Daten in der Eingabe vorhanden sind
// (bei dem Test wird gleich die Komponente Name eingelesen)

    while(gets(aktuell->Name) != NULL) {

// Komponente Kundennummer einlesen

        scanf("%d", &aktuell->KdNr);
```

```

// das Enter von scanf abfangen
// (was passiert wenn man getchar() vergisst ->
// siehe Skript 7 - 9 Programm a)

    getchar();

// schon mal ein neues Listenelement reservieren und
// die Komponente next (Zeiger) darauf zeigen lassen

    aktuell->next = (struct DATEN * ) malloc(sizeof(DATEN));

// das nächste, noch leere Element wird unser aktuelles
// ab hier wird also das neue Element durch aktuell referenziert

    aktuell = aktuell->next;
}

// auch falls das Ende der Daten erreicht ist, wurde noch ein
// neues Element angelegt -> dieses wird mit 'leeren' Werten
// gefüllt

    aktuell->KdNr = 0;
    aktuell->Name[0] = '\0';
    aktuell->next = NULL;

// Ausgabeteil

// zur Ausgabe müssen wir zum Anfang zurück

    aktuell = Start;

// solange noch Daten da sind

    while(aktuell->next != NULL) {

// ausgeben

        printf("\nName: %s\n", aktuell->Name);
        printf("\nKundennummer: %d\n", aktuell->KdNr);

// weiterrücken -> next zeigt auf das nächste Element

        aktuell = aktuell->next;
    }
}

```

```

/*
Skript S. 15 - 6

Liest Daten in eine doppelt verkettete Liste ein.
Diese wird nach Menge der Daten dynamisch erweitert.

Nachteil dieser Implementierung zur besseren Verständlichkeit:
Es wird ein Element zu viel angelegt.
*/

#include <stdio.h>          // Ein-/Ausgabe
#include <malloc.h>        // Speicherverwaltung (u.a. malloc())

// Definition eines Listenelements

struct DATEN {

// Datenteil

    int KdNr;
    char Name[30+1];

// Zeiger auf benachbarte Listenelemente

    struct DATEN *next, *prev;
};

void main ()
{

// Zeiger auf das Element,
// mit dem aktuell gearbeitet wird

    struct DATEN *aktuell = NULL;

// Anlegen eines neuen Listenelements (Achtung, es hat keinen Namen,
// daher wird Referenz (Pointer) benötigt)

    aktuell = (struct DATEN * ) malloc(sizeof(DATEN));

// da das erste Element keinen Vorgänger hat -> Zeiger ins Leere

    aktuell->prev = NULL;

// solange noch Daten in der Eingabe vorhanden sind
// (bei dem Test wird gleich die Komponente Name eingelesen)

    while(gets(aktuell->Name) != NULL) {

// Komponente Kundennummer einlesen

        scanf("%d", &aktuell->KdNr);

```

```

// das Enter von scanf abfangen
// (was passiert wenn man getchar() vergisst ->
// siehe Skript 7 - 9 Programm a)

    getchar();

// schon mal ein neues Listenelement reservieren und
// die Komponente next (Zeiger) darauf zeigen lassen

    aktuell->next = (struct DATEN * ) malloc(sizeof(DATEN));

// um uns den 'Rückweg' zu sichern, setzen wir den Vorgänger-
// zeiger des nächsten Elements auf das aktuelle

    aktuell->next->prev = aktuell;

// das nächste, noch leere Element wird unser aktuelles
// ab hier wird also das neue Element durch aktuell referenziert

    aktuell = aktuell->next;
}

// auch falls das Ende der Daten erreicht ist, wurde noch ein
// neues Element angelegt -> dieses wird mit 'leeren' Werten
// gefüllt

    aktuell->KdNr = 0;
    aktuell->Name[0] = '\0';
    aktuell->next = NULL;

// Ausgabeteil

// zur Ausgabe müssen wir zum Anfang zurück
// solange es einen Vorgänger gibt

    while(aktuell->prev != NULL)

// setze aktuell auf diesen Vorgänger

        aktuell = aktuell->prev;

// solange noch Daten da sind

    while(aktuell->next != NULL) {

// ausgeben

        printf("\nName: %s\n", aktuell->Name);
        printf("\nKundennummer: %d\n", aktuell->KdNr);

// weiterrücken -> next zeigt auf das nächste Element

        aktuell = aktuell->next;
    }
}

```