Prof. Dr.-Ing. J. Abke
Christian Kraus

Technische Hochschule Aschaffenburg
Informatik II

Übungsaufgabe 7 Einfach verkettete Liste

Lernziele:

- das Prinzip der einfach verketteten Liste verstehen
- die Funktionen für eine einfach verkettete Liste abstrakt anwenden können
- die Funktionen für die einfach verkettete Liste in der Sprache C implementieren können

Legende:

☞ Lektüre, die vor der Übung gelesen werden muss

✎ Fragen/Aufgaben, die vor der Übung zu bearbeiten sind

❖ Aufgaben, die in der Übung zu bearbeiten sind

7.1 Vorbereitungen zur Übung

Die Vorbereitungen und Fragen sind vor dem Übungstermin zu bearbeiten. Sie sind als Hilfen gedacht, um Ihnen die Aufgaben, die an den Übungsterminen selbst zu bearbeiten sind, zu erleichtern.

☞ Lesen Sie zur Vorbereitung der Übung das Kapitel „20 - Abstrakte Datentypen - Listen“ aus der Vorlesung Informatik II.
Weiterhin ist die Lektüre von Kapitel 21 in dem Buch „C von A bis Z“ von Jürgen Wolf zu empfehlen.

✎ Einleitende Fragen:

- Wie lauten die vier wesentlichen Operationen, die auf eine Liste angewendet werden können?

- Aus welchen Teilen besteht ein Listenelement einer einfach verketteten Liste?

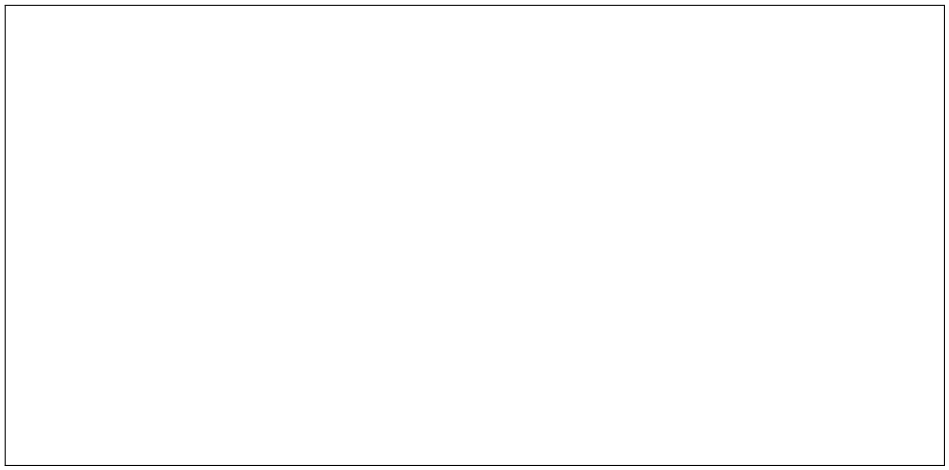
- Woran kann man bei einem Element einer einfach verketteten Liste erkennen, dass dieses das letzte Element der Liste ist?

Ein kleines Beispiel zum Aufwärmen

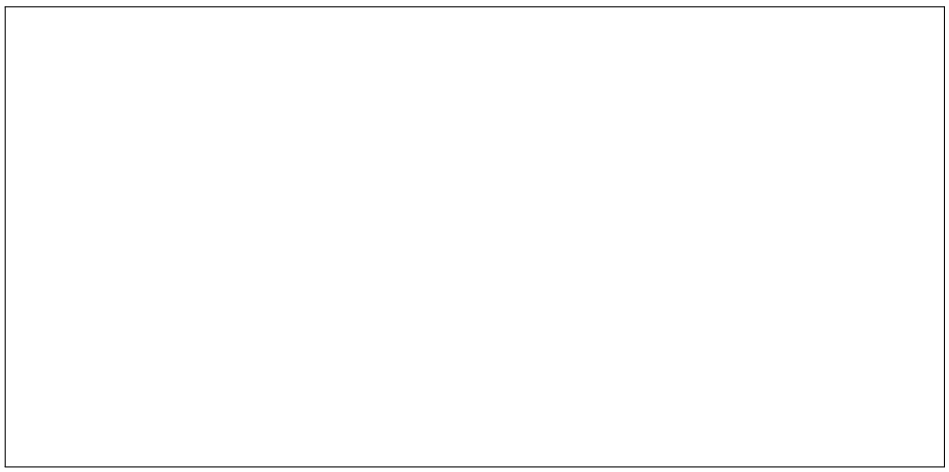
Schauen Sie sich die vorgegebene Headerdatei listen.h und deren Dokumentation ab 7.3 auf S. 6 an. Lesen Sie sich die Kommentare sorgfältig durch, die die Funktionen beschreiben. Die Liste wird über den Zeiger anfang_ptr *festgehalten*.

Skizzieren Sie die Liste mit allen Nutz- und Verwaltungsdaten, wenn...

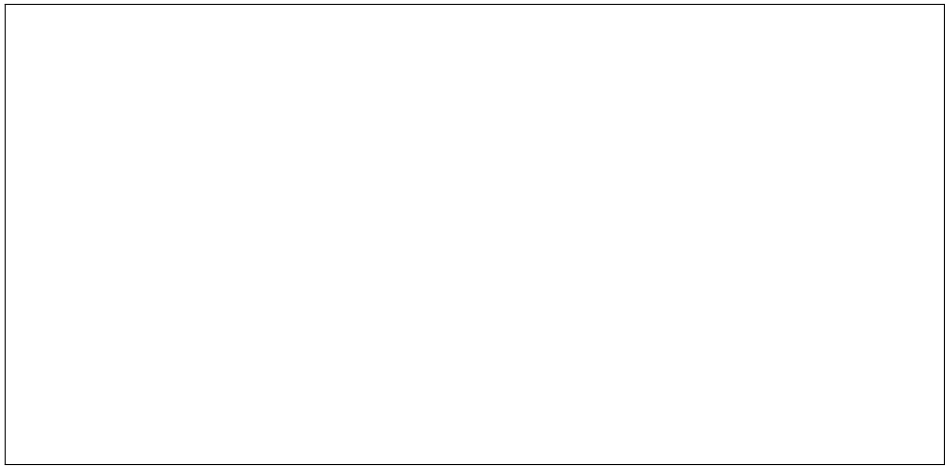
- ...das erste Element mit den Daten `"Vor", OUT_AD, 30, 500` eingefügt wurde.



- ...das Element mit den Daten `"Links", OUT_D, 30, 370` angefügt wurde.



- ... das Element mit den Daten `"Rechts", OUT_A, 30, 370` angefügt wurde.



7.2 Fahrsteuerung des EV3

❖Ziel dieser Übung ist es, die notwendigen Funktionen einer einfach verketteten Liste zu implementieren, die dann eine Fahrsteuerung auf dem EV3-Roboter ermöglicht.

Funktionsweise Fahrsteuerung

1. Die einzelnen Fahrbefehle (vorwärts, rückwärts, Kurve links, Kurve rechts) werden nacheinander über das Tastenkreuz ausgewählt, solange der Roboter still steht. Der Nutzer beendet die Eingabe der Fahrbefehle durch Druck auf die **BTNCENTER**-Taste.

2. Nach Ende der Fahrbefehlseingabe sollen die Befehle und deren Anzahl auf dem Display ausgegeben werden. Die Ausgabe geht automatisch über in den nächsten Punkt.

3. Abschließend fährt der Roboter die Fahrbefehle ab.

4. Das Programm ist anschließend mit der **BTNCENTER**-Taste vom Bediener zu beenden.

Implementierung der Fahrsteuerung

Die Fahrbefehle (vorwärts, rückwärts, Kurve links, Kurve rechts) sollen mit einer einfach verketteten Liste im Speicher abgelegt werden. Die zu übergebenden Daten pro Fahrbefehl sind bereits in der vorgegebenen Main-Funktion hinterlegt. Auch die Steuerung des Programms, wie vorstehend erläutert, ist bereits in der Datei `Uebung07_2.c` umgesetzt. Die beiden Dateien finden Sie auf Moodle. Fügen Sie diese entsprechend in Ihr Projekt ein. Ihre Aufgabe ist es, die entsprechenden **Funktionsdefinitionen** in der Datei `listen.c` zu realisieren. Eine Dokumentation der Listenelement- und Funktions-Deklarationen finden Sie in Abschnitt 7.3 ab Seite 6.

Hinweis

Um den String, der die Bezeichnung der Fahrtrichtung enthält, in die Liste zu kopieren, benutzen Sie bitte die Funktion strcpy aus string.h. Um die Funktion nutzen zu können, müssen Sie die Datei `string.h` in Ihre C-Datei per include-Anweisung **#include** <string.h> einbinden.
Die Syntax der Funktion lautet:

```
char *strcpy (char *destination, const char *source);
```

Die Funktion kopiert einen String, dessen Anfangsadresse Sie mit dem Zeiger source übergeben in den neuen Bereich mit der Anfangsadresse destination.

7.3 Listenelement- und Funktions-Deklarationen

7.3.1 Struktur fahrbefehl

```
struct fahrbefehl {
            char name[20];
            int motor;
            int power;
            int angle;
            struct  fahrbefehl * next;};
```

Folgende Daten sollen in einem Listenelement gespeichert werden:

- char **name** [20]

  Bezeichnung der Fahrtrichtung.
- int **motor**

  Angesteuerte Motoren.
- int **power**

  Motorleistung.
- int **angle**

  Drehwinkel der Motoren.
- struct **fahrbefehl** * **next**

  Zeiger auf naechstes Listenelement.

7.3.2 Funktion an_liste_anfuegen()

Fügt ein neues Listenelement am Ende der Liste hinzu.

```
void an_liste_anfuegen (
            struct  fahrbefehl ** start_ptr_ptr,
            char * name,
            int motor,
            int power,
            int angle )
```

Parameter

in	*start_ptr_ptr*	Zeiger auf Zeiger der Listenanfangsadresse
in	*name*	String enthaelt Fahrtrichtung
in	*motor*	Anzusteuernde Motoren
in	*power*	Leistung der Motoren
in	*angle*	Drehwinkel der Motoren

7.3.3 Funktion anzahl_listenelemente()

Listelemente zaehlen und auf dem Display ausgeben.

```
void anzahl_listenelemente (
            struct  fahrbefehl * cursor_ptr )
```

Parameter

in	*cursor_ptr*	Anfangsadresse der Liste

7.3.4 Funktion liste_abfahren()

Faehrt die einzelnen Befehle der Liste ab.

```
void liste_abfahren (
            struct  fahrbefehl * cursor_ptr )
```

Parameter

in	*cursor_ptr*	Anfangsadresse der Liste

7.3.5 Funktion liste_ausgeben()

Gibt den Inhalt aller Listenelemente auf dem Display aus.

```
void liste_ausgeben (
            struct  fahrbefehl * cursor_ptr )
```

Parameter

in	*cursor_ptr*	Anfangsadresse der Liste

Anhang

Uebung07_2.c

```
/*
 \file           Uebung07_2.c
 \author         kraus
 \brief          Main Funktion zur Uebung 7.2
 */

#include <ev3.h>
#include "listen.h"

/*!
    \brief Main-Funktion mit vordefinierten Werten fuer die
        Liste
    \return     Gibt bei Programmende 0 zurueck

*/
int main(void)
{
        struct fahrbefehl *anfang_ptr   = NULL; /**< Zeiger
            auf erstes Element **/
        int button = 0;


        while(button != BTNCENTER)

        {
                button = ButtonWaitForAnyPress(0);
                switch(button)
                {
                case BTNUP: TermPrintf( "Taste \"oben\"\n")
                    ;
                an_liste_anfuegen(&anfang_ptr,
                                "Vor", OUT_AD, 30, 500);
                break;

                case BTNDOWN: TermPrintf( "Taste \"unten\"\
                   n");
                an_liste_anfuegen(&anfang_ptr,
                                "Zurueck", OUT_AD, -30,
                                   500);
                break;
```

```
                case BTNLEFT: TermPrintf( "Taste \"links\"\
                   n");
                an_liste_anfuegen(&anfang_ptr,
                                  "Links", OUT_D, 30, 370);
                break;

                case BTNRIGHT: TermPrintf( "Taste \"rechts
                   \"\n");
                an_liste_anfuegen(&anfang_ptr,
                                  "Rechts", OUT_A, 30, 370);
                break;

                default: break;
                }
        }

        Ev3Clear();

        liste_ausgeben(anfang_ptr);
        anzahl_listenelemente(anfang_ptr);
        liste_abfahren(anfang_ptr);

        ButtonWaitForPress(BTNCENTER);

        return 0;
}
```