```
/***********************************
 * Filename      : UE08_liste.c     *
 * Created on    : Nov 23, 2018     *
 * Author        : Christian Zahner *
 ***********************************/

// Nach Vorgabe von Dr. Prof. Tempelmeier

#pragma compact_abi

#include "UART0.h"
#include "support_common.h"  // include peripheral declarations and more;
#include "uart_support.h"    // universal asynchronous receiver transmitter,
                             // (d.h. die serielle Schnittstelle)
#include "terminal_wrapper.h"
//#include "malloc_wrapper.h"

#include <stdlib.h>
#include <stdio.h>

#include "Intro.h"
#include "UE08_Liste.h"


char     msginput[]  = "\r\nZahl eingeben (0 fuer Ende): ";
char     msgfehler[] = "\nFehler bei malloc() !\n";
char     msgout[]    = "\nSotierte Liste lautet wie folgt: \n";
char     arrow[]     = "|-> ";

void     *anker = NULL;


int lstint(void){

 asm {

 schleife: // intialisiern von d0 und d7 test später erst

        // Zahl einlesen (Ende vereinfachend mit Wert 0)
        //----------------------------------------------

        clr.l   d0              // d0 mit 0 initialisieren
        pea     msginput        // msginput Adresse auf Stack
        jsr     TERM_WriteString
        adda    #4, SP          // Stack bereinigen

        // Zahl einlesen
        jsr     INOUT_ReadInt   // liest Int vom Terminal

        tst.w   d0              // Zerobyte test beendet Schleife
        beq     ausgabe
        move.l  d0, d7          // eingelesene Zahl in d7


        // Einzuhängendes Element aufbauen
        // -------------------------------
        // Mit malloc; malloc erwartet den Parameter im Register D0,
        // weil dafür "register_abi" gilt (nicht compact_abi)

        move.l  #6, d0          // für malloc groesse muss in d0 sein
        jsr     malloc          // bereits stellen von N Bytes auf Speicher
                                // Anzahl N steht in d0

        tst.l   a0              // a0 enthaelt Zeiger auf neues Objekt
```

```
        beq     fehler
        move.w  d7,(a0)             // Schreiben auf das wo a0 drauf zeigt
        clr.l   2(a0)               // Der hintere Bereich vom auf den Speicher der
                                    // vorher von malloc bereitgestellt wurde
                                    // null intsialisieren (hintern 4 Byte)
                                    // sind Pointer aufs naechste Element

    //
    //          a0 -->  ___________
    //                  |zahl|  0 |
    //

    // Organisation der Listeniteration
    //
    // a2 zeigt auf das aktuelle Element (oder 0 für Listenende)
    // a3 zeigt   a u f   d e n   Z e i g e r   im vorhergehenden Element
    //
    //                              a3         a2
    //                                \          \
    //  anker:                         \          \
    //  ______     ___________      _____V_____     V___________     ___________
    //  |    | --> |zahl|next| --> |zahl|next| --> |zahl|next| --> |zahl|  0 |
    //
    //
    //    Funtioniert
    //       + auch am Listenende (a3 zeigt auf 0, a2 ist 0)
    //       + auch bei Einfügen vor dem ersten Element
    //                         (a3 zeigt auf Anker, a2 auf das erste Element
    //       + auch bei leerer Liste (a3 zeigt auf Anker, Anker und a2 sind 0)


    // suchen der Einfügestelle
    //-------------------------

    // Iterator-Pärchen initialisieren

        lea     anker,  a3          // effektive Adresse vom Anker
                                    // (also Adresse vom Anker selbst)
        move.l  anker,  a2          // Adresse auf die Anker zeigt in a2

naechstes:  // weiterschalten bis a2 auf NULL zeigt

        tst.l   a2                  // test ob NULL bedeutet ist Listenende
        beq     gefunden            // wenn ende gefunden dann Sprung zu gefunden

        // cmp nur auf Long ext.l
        // da wir Int einlesen und diese nur 16 bit sind aber cmpare nur auf
        // long funtkioniert müssen wir die Dataregister noch erweitern
        move.w  (a2), d0            // Die Zahl auf die a2 aktuell zeigt
        ext.l   d0                  // von 2 Byte auf 4 Byte extension
        ext.l   d7                  // bereits vorher in Schleife Eingabe in d7 kopiert
        cmp.l   d0,d7

        ble     gefunden            // Springen auf Marke gefunden wenn die Eingegebene
                                    // Zahl kleiner bzw. gleich groß ist wie die Zahl
                                    // an der Stelle wo eingefügt werden soll

        // Weiterschalten des Doppeliteratoren
        lea     2(a2), a3           // wir wollen ja den Pointer von dem auf was a2 zeigt,
                                    // alternativ mit lea
        move.l  2(a2), a2           // wir wollen ja die nächste Zahl also auf
                                    // das was a3 jetzt zeigt
```

```
        bra     naechstes       // Es wird solange wiederholt bis das richtige
                                // Listenelemnt gefunden wurde

 // Einfügestelle gefunden: Objekt einhängen
 //-----------------------------------------
gefunden:
        move.l  a2, 2(a0)           // Hinteren 4 Byte des neune Elements
                                    // auf die Adresse von a2 zeigen lassen
        move.l  a0, (a3)            // Pointer auf das was a3 zeigt verbiegen
                                    // auf adfresse in a0 (neues Element)

        bra         schleife        // naechste Zahl

 // gesamte Liste ausgeben
 //-----------------------
ausgabe:
        jsr     TERM_WriteLn        // erstmal neue Zeile
        pea     msgout              // msgout Adresse auf Stack
        jsr     TERM_WriteString
        adda    #4, SP              // Stack bereinigen
        jsr     TERM_WriteLn        // erstmal neue Zeile
        move.l  anker, a2           // erneut Anfang der Liste in a2 laden
                                    // Diesmal genügt ein Iterator
ausschleife:
        tst.l   a2                  // test ob Listenende erreicht wurde
        beq     ende                // wenn Ende der Lsite erreicht Springe
                                    // auf Marke Ende

        // Ausgabe wie immer
        move.w  (a2), -(SP)         // Inhalt auf das wa a2 zeigt auf Stack
        jsr     INOUT_WriteInt
        adda    #2, SP              // Stack bereinigen

        pea     arrow               // masgarrow Adresse auf Stack
        jsr     TERM_WriteString
        adda    #4, SP              // Stack bereinigen

        //jsr       TERM_WriteLn    // Falls man die Liste
                                    // untereinander ausgegeben will

        //weiterschalten des Iterators
        move.l  2(a2), a2           // weiterschalten auf das nächste
                                    // Element der Liste
        bra     ausschleife

fehler:         //Fehler behandlung eigentlich nur Ausgabe
        pea     msgfehler           // msginput Adresse auf Stack
        jsr     TERM_WriteString
        adda    #4, SP              // Stack bereinigen
        bra     ende

ende:
        jsr     TERM_WriteLn        //neue Zeile
 }

TERM_WriteLn();                     // Nächste Zeile im Terminal

}
```