```c
/* list.c zu Aufgabe 1
 * (http://www.informatik.htw-dresden.de/%7Ebeck/C/PspCB1.html)
 *
 * -- Jan Losinski, 2008/04/14
 *  */

#include <stdlib.h>
#include "list.h"

/* Baut ein neues Listenelement und setzt gleich
 * die zugehoerigen Pointer.
 * Args:
 *   next    .. Nachfolger
 *   prev    .. Vorgaenger
 *   content .. Inhalt
 * Ret:
 *   Im Erfolgsfall einen Pointer auf das Element,
 *   sonst NULL
 * */
tCnct * CreateItem(tCnct * next, tCnct * prev, void * content){
  tCnct * item = NULL;
  if(( item = malloc(sizeof(tCnct)) ) != NULL){
    item->pItem  = content;
    item->pPrv   = prev;
    item->pNxt   = next;
  }
  return item;
}

/* Baut eine leere Liste, ohne jegliche Elemente.
 * Pointer werden alle mit NULL initialisiert.
 * Args:
 *   keine
 * Ret:
 *   Im Erfolgsfall einen Pointer auf die Liste,
 *   sonst NULL
 * */
tList * CreateList(void){
  tList * newList = NULL;
  if(( newList = malloc(sizeof(tList)) ) != NULL){
    newList->listHead = NULL;
    newList->listEnd  = NULL;
    newList->listCurr = NULL;
  }
  return newList;
}

/* Loescht eine Liste, sofern diese Leer ist.
 * Args:
 *   pList .. zu loeschende Liste
 * Ret:
 *   OK im Erfolgsfall, FAIL sonst.
 * */
int DeleteList(tList* pList){
  if (pList->listHead == NULL){
    free(pList);
    return OK;
  }
  return FAIL;
}

/* Fuegt ein neues Listenelement hinter dem aktuell
 * selektieren ein und selektiert es.
 * Behandelt auch den Fall, das die Liste noch leer
 * war (pList->listHead == NULL) oder das das Element
 * hinter dem eingefuegt werden soll das letzte in
 * der Liste war (pList->listCurr == pList->listEnd).
```

```
 * Args:
 *   pList    .. Die Liste auf der gearbeitet werden soll
 *   pItemIns .. Der Inhalt des neuen Elementes
 * Ret:
 *   OK im Erfolgsfall, FAIL sonst.
 * */
int InsertBehind  (tList* pList, void *pItemIns){
  tCnct * newItem = NULL;
  tCnct * next = NULL;
  if (pList->listCurr != NULL){
    next = (pList->listCurr)->pNxt;
  }
  if ((newItem = CreateItem(next, pList->listCurr, pItemIns)) != NULL){
    if (pList->listHead == NULL){
      pList->listHead = newItem;
      pList->listEnd = newItem;
    } else {
      if (pList->listCurr == pList->listEnd) {
        pList->listEnd = newItem;
      }
    }
    pList->listCurr = newItem;
    return OK;
  } else {
    return FAIL;
  }
}

/* Fuegt ein neues Listenelement vor dem aktuell
 * selektieren ein und selektiert es.
 * Behandelt auch den Fall, das die Liste noch leer
 * war (pList->listHead == NULL) oder das das Element
 * vor dem eingefuegt werden soll das erste in
 * der Liste war (pList->listCurr == pList->listHead).
 * Args:
 *   pList    .. Die Liste auf der gearbeitet werden soll
 *   pItemIns .. Der Inhalt des neuen Elementes
 * Ret:
 *   OK im Erfolgsfall, FAIL sonst.
 * */
int    InsertBefore  (tList* pList, void *pItemIns){
  tCnct * newItem = NULL;
  tCnct * prev = NULL;
  if (pList->listCurr != NULL){
    prev = (pList->listCurr)->pPrv;
  }
  if ((newItem = CreateItem(pList->listCurr, prev, pItemIns)) != NULL){
    if (pList->listHead == NULL){
      pList->listHead = newItem;
      pList->listEnd = newItem;
    } else {
      if (pList->listCurr == pList->listHead) {
        pList->listHead = newItem;
      }
    }
    pList->listCurr = newItem;
    return OK;
  } else {
    return FAIL;
  }
}

/* Fuegt ein neues Listenelement an den Anfang der
 * Liste ein und selektiert es.
 * Dies tut es, indem es das erste Element der Liste
 * selektiert und das neue Element vor dem selektiertem
 * einfuegen laesst
```

```
 * Args:
 *   pList    .. Die Liste auf der gearbeitet werden soll
 *   pItemIns .. Der Inhalt des neuen Elementes
 * Ret:
 *   OK im Erfolgsfall, FAIL sonst.
 * */
int InsertHead (tList* pList, void *pItemIns){
  pList->listCurr = pList->listHead;
  return InsertBefore(pList, pItemIns);
}

/* Fuegt ein neues Listenelement an das Ende der
 * Liste ein und selektiert es.
 * Dies tut es, indem es das letzte Element der Liste
 * selektiert und das neue Element nach dem
 * selektiertem einfuegen laesst
 * Args:
 *   pList    .. Die Liste auf der gearbeitet werden soll
 *   pItemIns .. Der Inhalt des neuen Elementes
 * Ret:
 *   OK im Erfolgsfall, FAIL sonst.
 * */
int InsertTail (tList* pList, void *pItemIns){
  pList->listCurr = pList->listEnd;
  return InsertBehind(pList, pItemIns);
}

/* Loescht das aktuell selektierte Element aus der
 * Liste.
 * Behandelt auch den fall, das das geloeschte
 * Element das erste, das letzte oder das einzigste
 * der Liste war.
 * Die Pointer der umliegenden Elenmente werden
 * neu gesetzt.
 * Args:
 *   pList .. Die Liste aus der geloescht wird
 * Ret:
 *   nichts
 * */
void RemoveItem (tList* pList){
  if (pList->listCurr != NULL){
    tCnct * curr = pList->listCurr;
    tCnct * next = curr->pNxt;
    tCnct * prev = curr->pPrv;
    if (prev != NULL){
      prev->pNxt = next;
      pList->listCurr = prev;
    } else {
      pList->listHead = next;
      pList->listCurr = NULL;
    }
    if (next != NULL){
      next->pPrv = prev;
      pList->listCurr = next;
    } else {
      pList->listEnd = prev;
    }
    free(curr);
  }
}

/* Gibt den Inhalt des momentan selektierten
 * Elementes zurueck
 * Args:
 *   pList .. Die Liste, in der das Element ist
 * Ret:
 *   Der Inhalt des Elementes oder NULL im
```

```
 *   Fehlerfall.
 * */
void* GetSelected    (tList* pList){
  if (pList->listCurr != NULL){
    return (pList->listCurr)->pItem;
  } else {
    return NULL;
  }
}

/* Gibt den Inhalt des ersten Elementes zurueck
 * und selektiert es
 * Args:
 *   pList .. Die Liste, in der das Element ist
 * Ret:
 *   Der Inhalt des Elementes oder NULL im
 *   Fehlerfall.
 * */
void* GetFirst       (tList* pList){
  pList->listCurr = pList->listHead;
  return GetSelected(pList);
}

/* Gibt den Inhalt des letzten Elementes zurueck
 * und selektiert es
 * Args:
 *   pList .. Die Liste, in der das Element ist
 * Ret:
 *   Der Inhalt des Elementes oder NULL im
 *   Fehlerfall.
 * */
void* GetLast        (tList* pList){
  pList->listCurr = pList->listEnd;
  return GetSelected(pList);
}

/* Gibt den Inhalt des naechsten Elementes zurueck
 * und selektiert es
 * Args:
 *   pList .. Die Liste, in der das Element ist
 * Ret:
 *   Der Inhalt des Elementes oder NULL im
 *   Fehlerfall.
 * */
void* GetNext        (tList* pList){
  if (pList->listCurr != NULL){
    pList->listCurr = (pList->listCurr)->pNxt;
  }
  return GetSelected(pList);
}

/* Gibt den Inhalt des vorhergehenden Elementes
 * zurueck und selektiert es
 * Args:
 *   pList .. Die Liste, in der das Element ist
 * Ret:
 *   Der Inhalt des Elementes oder NULL im
 *   Fehlerfall.
 * */
void* GetPrev        (tList* pList){
  if (pList->listCurr != NULL){
    pList->listCurr = (pList->listCurr)->pPrv;
  }
  return GetSelected(pList);
}

/* Gibt den Inhalt des N-ten Elementes Zurueck,
```

```
 * indem es es die Liste bis zu diesem durchlauft.
 * Sollte die Liste kuerzer als N sein, so wird
 * NULL zurueck gegeben
 * Args:
 *   pList .. Die Liste, in der das Element ist
 *   Idx   .. Der Index N, an dem das Element zu
 *            finden ist.
 * Ret:
 *   Der Inhalt des Elementes oder NULL im
 *   Fehlerfall bzw. wenn der Index groesser als
 *   die Liste.
 * */
void* GetIndexed   (tList* pList,int Idx){
  tCnct * walker = pList->listHead;
  int i;
  for (i = 0; i < Idx; i++){
    if (walker->pNxt != NULL){
      walker = walker->pNxt;
    } else {
      return NULL;
    }
  }
  pList->listCurr = walker;
  return GetSelected(pList);
}

/* Fuegt ein Element mit einer als Pointer
 * uebergebenen Funktion in die Liste ein.
 * Args:
 *   pList .. Die Liste
 *   pItem .. Der Elementinhalt
 *   fcmp  .. Der Funktionspointer
 * Ret:
 *   Das eingefuegte oder momentan selektierte
 *   Element
 * */
void* addItemToList (tList* pList, void * pItem, int(*fcmp)(void*pItList,void*pItNew)){
  fcmp(pList,pItem);
  return pList->listCurr;
}

// Nur zum Test...
int main (){
  return 0;
}
```