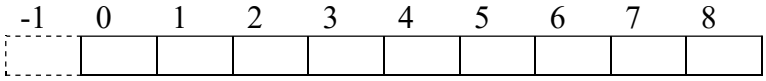
C-ÜBUNGSAUFGABEN VERKETTETE LISTEN 1

Bemerkungen:
B1)
Wenn man die verkettete Liste als Feld auffasst, in dem pos die Stelle des letzten Elements bezeichnet (wenn das Feld die Länge 0 hat ist diese Stelle = -1), dann kann man die Stelle angeben, wo der Wert eingefügt werden soll.
Nummerierung der Zellen für ein Feld der Länge 9 (das eine verkettete Liste simuliert):



Für pos muß gelten: -1<=pos <=Stelle des letzten Elements.
Wenn also in einer der folgenden Methoden der Parameter pos vorkommt, dann wird so ein Feld betrachtet.

1) Einfach verkettete Listen

Implementieren Sie die wie folgt dokumentierten Funktionen für einfach verkettete Listen
In main() sollen diese Funktionen ausführlich (z.B. mit Hilfe von Schleifen) getestet werden.
Eine verkette Liste wird auch kurz nur mit Liste bezeichnet.

Datenstruktur für eine einfach verkettet Liste:

```
struct dtelement{
  int zahl;
  struct dtelement *next;
};
```

```
/*************************************************************/
/**                                                         **/
/**  struct dtelement *appendElement1(struct dtelement *last, **/
/**                                   int value)            **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *last : Adresse des letzten Listenelements
  (i) int value : Zahl, die an die Liste angefügt werden soll.

Return:
  (o) die Adresse des (neuen) letzten Listenelements.
      NULL, falls kein Speicher reserviert werden kann.

Beschreibung:
  Fügt ein Element an das letzte Listenelement einer Liste an und
  gibt die Adresse des (neuen) letzten Listenelemenst
  zurück.
  Parameter last gleich NULL, bedeutet daß eine Liste erzeugt wird
*/
```

```
/*************************************************************/
/**                                                         **/
/**  struct dtelement *appendElement2(struct dtelement *list,  **/
/**                                   int value)            **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i) int value : Zahl, die an die Liste angefügt werden soll.

Return:
  (o) die Adresse des (neuen) letzten Listenelements.
      NULL, falls kein Speicher reserviert werden kann.

Beschreibung:
  Fügt ein Element an das letzte Listenelement einer Liste an.
  Da als Parameter nur die Anfangsadresse übergeben wird, muß die
  Funktion selbst das letzte Element der Liste berechnen
  (sich vom Anfang zum Ende hangeln).
  Parameter list gleich NULL, bedeutet daß eine Liste erzeugt wird
*/


/*************************************************************/
/**                                                         **/
/**  void printList(struct dtelement *list)                 **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste

Return:
  nichts

Beschreibung:
  Gibt die Daten, also Zahlen (nicht Adressen) aller Elemente der
  Liste auf dem Bildschirm aus.
*/
```

```
/*************************************************************/
/**                                                         **/
/**  struct dtelement *gradeRight(struct dtelement *list,   **/
/**                               int value)                **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i) int value : Zahl, die in die Liste eingefügt werden soll.

Return:
  (o) die Adresse des (neuen) letzten Listenelements.
      NULL, falls kein Speicher reserviert werden kann.

Beschreibung: (Rechts einsortieren)
  Fügt den Wert rechts von der Stelle ein, wo value das 1. Mal vom
  Wert her größer (oder gleich) ist als das dort sich befindliche
  Feldelement.
  Ansonsten wird es rechts von der Stelle -1 eingefügt.
  Wenn nur diese Funktion verwendet wird, um Werte einzufügen,
  ergibt dies eine aufsteigend sortierte Liste.

Beispiel:
  verkettete Liste:   4   5   9   1   3
  Rechts einsortieren der Zahl 8 ergibt:
  verkettete Liste:   4   5   8   9   1   3
*/


/*************************************************************/
/**                                                         **/
/**  struct dtelement *gradeLeft(struct dtelement *list,    **/
/**                              int value)                 **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i) int value : Zahl, die in die Liste eingefügt werden soll.

Return:
  (o) die Adresse des (neuen) letzten Listenelements.
      NULL, falls kein Speicher reserviert werden kann.

Beschreibung: (Links einsortieren)
  Fügt das Element zahl links von der Stelle ein, wo zahl das 1. Mal
  vom Wert her größer (oder gleich) ist als das dort sich
  befindliche Feldelement.
  Ansonsten wird es an das Ende der Liste angefügt.
  Wenn nur diese Funktion verwendet wird, um Werte einzufügen,
  ergibt dies eine absteigend sortierte Liste.

Beispiel:
  verkettete Liste::   4   5   9   1   3
  Links einsortieren der Zahl 2 ergibt:
  verkettete Liste:   4   5   9   2   1   3
*/
```

```
/*************************************************************/
/**                                                         **/
/**  struct dtelement *insertRight(struct dtelement *list,  **/
/**                                int pos, int value)      **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i) int pos   : Stelle, an der eingefügt werden soll.
  (i) int value : Zahl, die in die Liste eingefügt werden soll.

Return:
  (o) die Adresse des (neuen) letzten Listenelements.
      NULL, falls kein Speicher reserviert werden kann.

Beschreibung: (Links einsortieren)
  Für pos muß gelten:  -1<=pos <=Stelle des letzten Elements.
  Der Wert value wird rechts von pos, also an der Stelle pos+1
  eingefügt (nicht überschrieben)
  Vorher werden die entsprechenden Zahlen noch nach rechts
  verschoben.

Beispiel:
  verkettete Liste:   7   5   9   3
  Einfügen der Zahl 8 rechts von pos = 2 ergibt:
  verkettete Liste:   7   5   9   8   3
*/


/*************************************************************/
/**                                                         **/
/**  void deleteList(struct dtelement *list)                **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste

Return:
  nichts

Beschreibung:
  löscht alle Elemente der verketteten Liste, indem jeder pro
  Element reservierte Speicher wieder freigegeben wird.
*/
```

```
/*************************************************************/
/**                                                         **/
/**  void deleteElement1(struct dtelement *element,         **/
/**                      struct dtelement *prec)            **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *element : Adresse des zu löschenden
                                  Elements
  (i) struct dtelement *prec : Adresse des vorigen (benachbarten)
                               Elemets von "element"

Return:
  nichts

Beschreibung:
  Löscht das Listenelement "element" einer einfach verketteten
  Liste. Da die Liste nur einfach verkettet ist, braucht die
  Funktion noch die Adresse von "prec", also dem vorigen Element
  von "element", damit prec mit dem Nachfolger von "element"
  verlinkt werden kann.
  Bei einer doppelt verketteten Liste wäre dies nicht nötig.
*/


/*************************************************************/
/**                                                         **/
/**  void deleteElement2(struct dtelement *prec)            **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *prec : Adresse des vorigen (links
                               benachbarten) Elements.

Return:
  nichts

Beschreibung:
  Löscht nicht das Listenelement "prec" einer einfach verketteten
  Liste, sondern das Element rechts dieses Elements.
  Da die Liste nur einfach verkettet ist, kann die Funktion
  prec mit dem übernächsten verlinken bzw. das nächste löschen.
  Bei einer doppelt verketteten Liste wäre dies nicht nötig:
  man müsste nur die Adresse des zu löschendem Elements angeben.
*/
```

```
/*************************************************************/
/**                                                         **/
/**  void deleteLastElement(struct dtelement *element)      **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *element : Anfangsadresse der Liste

Return:
  nichts

Beschreibung:
  Löscht das letzte Listenelement der Liste.
*/


/*************************************************************/
/**                                                         **/
/**  int deleteValue(dtelement *list, int value)            **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i) int value : Zahl, die in die Liste gelöscht werden soll.

Return:
   -1: Zahl wurde nicht gefunden, also konnte nicht gelöscht werden
   0: Zahl wurde gefunden, also auch gelöscht.

Beschreibung:
  Löscht die Zahl "value" in einer Liste.
  Wenn mehrere gleiche Zahlen in der Liste vorkommen, wird die erste
  vorkommende Zahl gelöscht.
*/
```

```
/************************************************************/
/**                                                        **/
/**  void deleteAt(dtelement *list, int pos)               **/
/**                                                        **/
/*#***********************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i) int pos: Stelle, an der das Element gelöscht wird.

Return:
   -1: Zahl wurde nicht gefunden, also konnte nicht gelöscht werden
    0: Zahl wurde gefunden, also auch gelöscht.

Beschreibung:
  Für pos muß gelten:  0 <= pos <=Stelle des letzten Elements.
  Löscht die Zahl an der Stelle pos.

Beispiel:
  verkettete Liste:   4   5   9   1   3
  Lösche Element an der Stelle 2
  verkettete Liste:   4   5   1   3


*/
/************************************************************/
/**                                                        **/
/**  int searchValue(struct dtelement *list, int value)    **/
/**                                                        **/
/*#***********************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i) int value : Zahl, die in der Liste gesucht werden soll.

Return:
  -1: Zahl value wurde nicht gefunden
  0: Zahl gefunden

Beschreibung:
  Sucht eine Zahl in einer Liste und gibt das Ergebnis zurück:
  0: gefunden,  -1: nicht gefunden.
*/


/************************************************************/
/**                                                        **/
/**  void sortList(struct dtelement *list, int modus)      **/
/**                                                        **/
/*#***********************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i)    int modus: (0: aufsteigend, 1: absteigend)

Return:
  nichts

Beschreibung:
  Sortiert eine Liste aufsteiegend (modus=0) oder
  absteiegend (modus=1).
*/
```

```
/*************************************************************/
/**                                                         **/
/**  void change(struct dtelement *list, int valueOld,      **/
/**              int valueNew)                              **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste
  (i) int valueOld: die zu ersetzende Zahl
  (i) int valueNew: diese neue Zahl ersetz die alte Zahl

Return:
   -1: oldValue wurde nicht gefunden, konnte also nicht ersetzt
       werden.
   0: oldValue wurde gefunden, also auch ersetzt.

Beschreibung:
  Ersetzt das erste Vorkommen der Zahl oldValue durch die neue Zahl
  newValue.
*/


/*************************************************************/
/**                                                         **/
/**  int deleteDuplicate(struct dtelement *list)            **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *list : Anfangsadresse der Liste

Return:
   Anzahl der Duplikate in der Liste

Beschreibung:
  Löscht die Elemente einer Liste die mehr als 1 Mal vorkommen, so
  daß jedes Element genau ein Mal in der Liste vorkommt.

Beispiel:
  verkettete Liste:   7   5   7   9   5   8   9
  Duplikate löschen
  verkettete Liste:   7   5   9   8
*/
```

Bemerkungen:

B1)

In main() sollen alle obigen Funktionen ausführlich (z.B. mit Hilfe von Schleifen) getestet werden.

B2)

Weitere, selbst erfundene Funktionen implementieren.

B3)

Überlegen Sie, ob es auch noch Sinn macht, die Adressen des Listenanfangs und Listenendes in struct dtelement zu speichern.

2) Doppelt verkettete Listen

Implementieren Sie die oben dokumentierten Funktionen, jetzt aber für doppelt verkettete Listen. Passen Sie die Parameter bzw. die Beschreibung - falls mötig - entsprechend an.
In main() sollen diese Funktionen ausführlich (z.B. mit Hilfe von Schleifen) getestet werden.
Eine verkette Liste wird auch kurz nur mit Liste bezeichnet.

Ein paar Beispiele:

```
/*************************************************************/
/**                                                         **/
/**  void printList(struct dtelement *element)              **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *element : Adresse eines Listenelements

Return:
  nichts

Beschreibung:
  Gibt die Daten, also Zahlen (nicht Adressen) aller Elemente der
  doppelt verketteten Liste auf dem Bildschirm aus, wobei element
  nicht die Anfangsadresse der Liste sein muß, sondern die Adresse
  irgend eines Elements der Liste ist.
  Die Elemente der Liste werden vom Listenanfang bis Listenende
  hintereinander ausgegeben. Intern muß die Funktion dazu von
  "element" ausgehend den Listenanfang bestimmen.
*/


/*************************************************************/
/**                                                         **/
/**  void deleteElement(struct dtelement *element)          **/
/**                                                         **/
/*#************************************************************/
/*
Parameter:
  (i) struct dtelement *element : Adresse des zu löschenden Elements

Return:
 nichts

Beschreibung:
  Löscht das Listenelement "element" einer doppelt verketteten
  Liste.
*/
```

3) Warteschlange

Implementieren Sie eine Warteschlange (z.B. bei einem Drucker) als spezielle verkettete Liste, die dem Prinzip folgt: First in - First out.
Implementieren Sie dazu noch (siehe oben) die entsprechenden Funktionen (anfügen, usw.) .

4) Stack

Implementieren Sie einen Stapel (Stack), als spezielle verkettete Liste, der dem Prinzip folgt: Last in - First out.
Implementieren Sie dazu noch (siehe oben) die entsprechenden Funktionen (anfügen, usw.) .

5) Ringpuffer

Man hat einen Speicher mit N Elementen (von 0 bis N-1 insiziert). Das dem N-1 - ten Element folgende ist dann wieder das Element 0. D.h. das 0. Element wird überschrieben, danach 1. Element , usw. Implementieren Sie einen Ringpuffer.

6) Binärbaum

Ein Binärbaum hat eine Wurzel. Von jedem Knoten gehen 2 weitere Knoten ab. Damit kann man z.B. Zahlen einsortieren. Im linken Knoten werden Zahlen kleiner gleich und im rechten Knoten Zahlen größer als der Elternknoten abgespeichert.

								100								
				30								200				
		15				40				150				300		
														400		

Implementieren Sie eine Funktion, die Zahlen in einen Binärbaum einsortiert bzw. sucht.