Übung Listen

```
typedef int ELEMENTTYP;
struct node {
   ELEMENTTYP info;
   node * next;
};
```

Implementieren Sie die im folgenden beschriebenen Funktionen auf Basis der gegebenen Datenstruktur node! Überlegen Sie sich geeignete Parameter und versuchen Sie, einmal geschriebene Funktionen weitestgehend wiederzuverwenden! Die Verwendung globaler Variablen ist untersagt!

createList() // Erzeugt eine neue Liste

isEmpty() // testet, ob die übergebene Liste leer ist

writeList() // gibt alle Listenelemente sequentiell aus

insertAtHead() // fügt ein übergebenes Element am Listenkopf ein

insertAtTail() // fügt ein übergebenes Element am Listenende ein

search() // liefert Zeiger auf gesuchtes Element oder NULL-Zeiger
// Implementieren Sie eine iterative und eine rekursive Variante!

insertSorted() // fügt ein neues Listenelement geordnet ein
// Implementieren Sie eine iterative und eine rekursive Variante!

Übung Listen

```
typedef int ELEMENTTYP;
struct node {
   ELEMENTTYP info;
   node * next;
};
```

Implementieren Sie die im folgenden beschriebenen Funktionen auf Basis der gegebenen Datenstruktur node! Überlegen Sie sich geeignete Parameter und versuchen Sie, einmal geschriebene Funktionen weitestgehend wiederzuverwenden! Die Verwendung globaler Variablen ist untersagt!

deleteFirst() // löscht das erste Listenelement

delete() // löscht **das erste** Listenelement mit dem vorgegebenen Wert

// Implementieren Sie eine iterative und eine rekursive Variante!

mirror() // spiegelt die übergebene Liste

getMin() // liefert minimalen Elementwert der Liste. Überlegen Sie sich, wie

// man den Fall der leeren Liste abfangen könnte

sortList() // gibt eine aufsteigend sortierte Liste zurück