9. Übungsblatt
Thema: verkettete Listen, abstrakte Datentypen

Aufgabe 35
Selbstanordnende Listen
Elemente, auf die häufig zugegriffen wird, sollen möglichst weit vorn plaziert werden. Nach jedem Zugriff wird die Liste so verändert, daß eine künftige Suche nach diesem Element schneller geht (Beachte: Diese Listen sind **nicht** nach Schlüsselwerten sortiert!)

Es gibt verschieden Strategien:

a) *Move-to-front (MF)-Regel:* Das gefundene Element wird an den Kopf der Liste gebracht, die relative Anordnung der übrigen Elemente bleibt unverändert.

b) *Transpose (T) - Regel:* Der Tausch erfolgt mit dem Vorgänger des gefundenen Elementes.

c) *Frequency Count (FC)-Regel:* Ordne jedem Element einen Häufigkeitszähler zu, der die Anzahl der Zugriffe auf das jeweilige Element speichert. Die Liste wird nach jedem Zugriff in absteigender Reihenfolge des Häufigkeitszählers sortiert.

Entwerfen Sie für alle drei Strategien Algorithmen. Bei der Implementation nehmen Sie als Dateninhalt ganze Zahlen an.

Aufgabe 36
Schreiben Sie ein Programm für **doppelt verkettete** Listen (mit Zeiger auf Nachfolger **und** Vorgänger) mit folgenden Eigenschaften:

a) Die Listenstruktur soll so angelegt sein, daß als Dateninhalt Zeiger (auf beliebige Datenstrukturen) verwaltet werden können.
b) Außerdem ist ein Cursor (das ist die Position in der Liste, auf die sich dann z. B. Einfüge- und Löschoperationen beziehen) einzuführen.

Für die Liste sind folgende Operationen zu definieren:

- Erzeugen der Liste
- Löschen der gesamten Liste
- Element einfügen
- Element entfernen
- aktuelle Position des Cursors (vorwärts und rückwärts) ändern

Schreiben Sie einen geeigneten Testrahmen.

Aufgabe 37
Schreiben Sie eine Funktion *MAP*, die eine als Parameter gegebene Funktion auf alle Elemente einer Liste anwendet:

```
void MAP (LISTPTR list, void funct (LISTPTR le))
{
        ...
}
```

Aufgabe 38

Der *Stack* (Stapel, Keller oder Kellerspeicher) ist ein abstrakter Datentyp. Bei ihm wird das Einfügen oder Entfernen eines Elementes nach dem Prinzip LIFO (Last In, First Out) realisiert. Allgemein bekannte Operationen auf dem Stack sind:

CREATE	-	erzeugt einen leeren Stack
PUSH	-	legt ein Element auf den Stack
POP	-	liefert das oberste Element des Stacks und
EMPTY	-	prüft, ob Stack leer ist.

a) Diskutieren Sie mögliche Datenstrukturen für die Implementation der Operationen auf dem *Stack*

b) Entwerfen Sie ein Programm, das die Berechnung von arithmetischen Ausdrücken für die vier Grundrechenarten mittels *Stack* realisiert.

Hinweis: Die Berechnung wird in zwei Schritte aufgeteilt:

- Umwandlung eines Infix-Ausdruckes in die Postfix-Notation
- Auswertung der Postfix-Notation.