Übungsaufgaben zur Klausurvorbereitung
Algorithmen und Datenstrukturen

Aufgabe 1. (Iterative Funktion)

Unter einer ***geometrischen Folge*** versteht man eine regelmäßige mathematische Zahlenfolge bei der das Verhältnis zweier benachbarter Folgenglieder (a_i / a_{i-1}) konstant ist. Beispiel:

1, 1/2, 1/4, 1/8, ...

Somit kann das i-te Glied der Folge mit der folgenden rekursiven Formel bestimmt werden:

$a_0 = 1$
$a_i = a_{i-1} \cdot q$ für $i > 0$

wobei im Beispiel q = 0.5 gewählt wurde.

Entwickeln Sie eine ***iterative*** Funktion zur Berechnung der **Summe der ersten n Glieder** der oben gezeigten geometrischen Folge (sogenannte *geometrische Reihe*) mit q = 0.5. Die Funktion soll n als Übergabeparameter bekommen und das Ergebnis als Rückgabewert liefern, z.B. 1 + 0.5 = 1.5 für n = 2 oder 1 + 0.5 + 0.25 = 1.75 für n = 3. Der Prototyp der Funktion soll wie folgt aussehen:

```
double geoReiheIter (int n);
```

Aufgabe 2. (Rekursive Funktion)

Entwickeln Sie eine ***rekursive*** Funktion zur Berechnung der **Summe der ersten n Glieder** einer geometrischen Folge (sogenannte *geometrische Reihe*) mit q = 0.5. Die Funktion soll geeignete Übergabeparameter haben und wieder das Ergebnis als Rückgabewert liefern.

Aufgabe 3. (Einfach verkettete Liste) → Klausuraufgabe SS 2007

Bei einer zyklischen Liste weist der Verkettungszeiger des letzten Elementes wieder auf das erste zurück (s. Abbildung 4a. auf der nächsten Seite).
Gegeben sei eine solche einfach verkettete, zyklische Liste (**ohne** Pseudo-Kopfelement), deren Elemente folgende Struktur aufweisen:

```
typedef struct ListElmt_ {
        int                 data;
        struct ListElmt_    *next;
} ListElmt;
```

Die Elemente sind, wie in der Abbildung 4a. gezeigt, der Reihe nach durchnummeriert, d.h. das erste Element hat den Datenwert 1, das zweite den Wert 2, usw.

Entwickeln Sie ein Unterprogramm

```
int josephus (ListElmt *Kreis, int m);
```

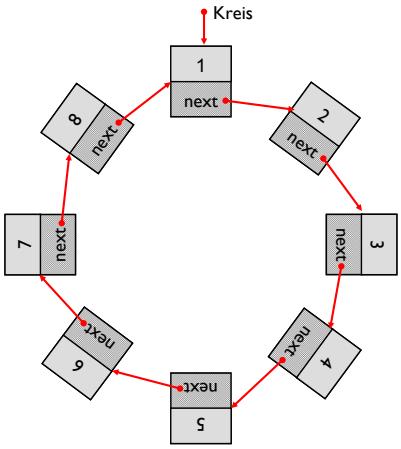
welches das sogenannte *Josephus-Problem* löst: Beginnend beim Element, auf das der übergebene Zeiger `Kreis` zeigt, wird losgezählt. Das m-te Element (`m` ist ein Übergabeparameter der Funktion) wird aus der Liste entfernt, und die Liste wieder korrekt verkettet. Anschließend beginnt das Zählen beim Nachfolger des entfernten Elementes von neuem und wieder wird das m-te Element entfernt. Der Vorgang wiederholt sich so oft, bis nur noch ein Element übrig ist. Der Datenwert dieses letzten verbleibenden Elementes soll vom Unterprogramm zurückgegeben werden. Bei Aufruf mit einem unzulässigen Wert für m soll das Programm hingegen -1 zurückgeben.

Sie dürfen zur Vereinfachung davon ausgehen, dass das Programm nur für Listen mit mindestens einem Element angewendet wird. Achten Sie aber auf die korrekte Freigabe von nicht mehr benötigtem Speicherplatz.

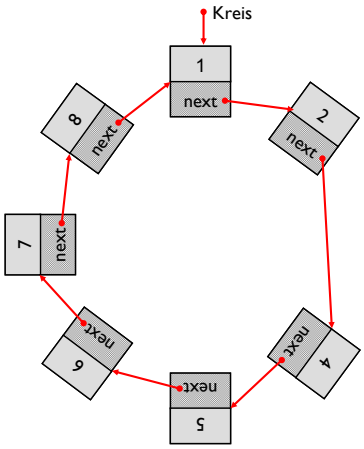
Geben Sie das Unterprogramm in C an.

Die Abbildungen 4a. bis 4e. (auf der nächsten Seite) verdeutlichen an einem Beispiel, wie sich die Datenstruktur für m=3 entwickeln soll.

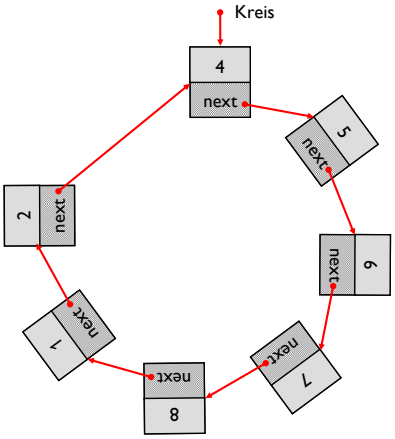
Illustration: nächste Seite



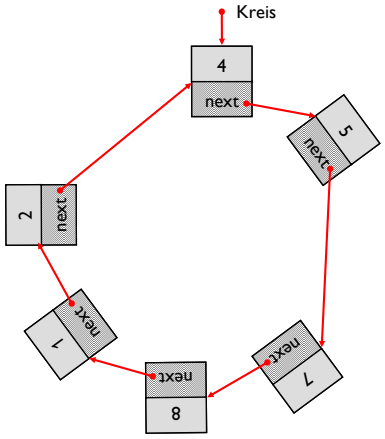
4a. Ausgangspunkt



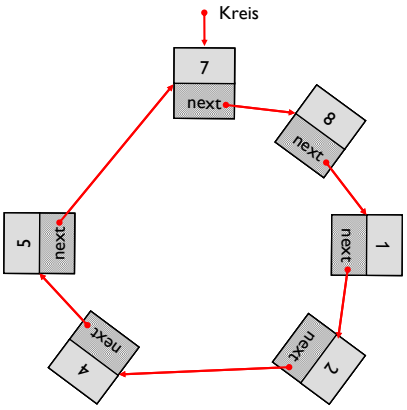
4b. Entfernen des 3. Elementes



4c. Neuer Ausgangspunkt



4d. Entfernen des 3. Elementes



4e. Neuer Ausgangspunkt

usw. ...

Aufgabe 4. (Listen)

Hinweis: Die Implementierung dieses Programms ist eine der Laboraufgaben.

Gegeben sei eine einfach verkettete Liste, deren Elemente folgende Struktur aufweisen:

```
typedef struct ListElmt_ {
        int                 data;
        struct ListElmt_    *next;
} ListElmt;
```

Das erste physikalische Listenelement diene lediglich als Kopfelement, dessen Nutzdatenwert (data) nicht verwendet wird und dessen next-Zeiger auf das erste logische Listenelement zeigt. Der next-Zeiger des letzten Listenelementes ist NULL.

Entwickeln Sie ein Unterprogramm

```
ListElmt* combineLists (ListElmt  *head1, ListElmt *head2);
```

welches zwei Listen im Reißverschlußverfahren zu einer einzigen zusammenfügt. Das Programm bekommt als Übergabeparameter Zeiger auf die Kopfelemente der beiden Listen und liefert als Rückgabewert einen Zeiger auf das Kopfelement der Ergebnisliste. Es muss ferner die folgenden Bedingungen erfüllen:
Die Implementierung soll **keine** Rekursion verwenden und keine neuen Listenelemente allokieren. Sie muss außerdem mit konstant großem Hilfsspeicher (unabhängig von der Anzahl der Elemente in den Listen) auskommen und höchstens lineare Laufzeit O(n+m) haben (wobei n und m die Längen der beiden Listen sind).

Aufgabe 5. (Sortieren)

(**Hinweis:** MergeSort wurde in der Vorlesung in diesem Jahr nicht behandelt. Daher handelt es sich eher um eine Zusatzaufgabe für Interessierte.)

a. Sortieren Sie die Zahlenfolge

4 2 17 21 8 13 9 1

mit Hilfe von Merge Sort und skizzieren Sie die Zwischenlösung in geeigneter Weise **nach** jeder Verschmelzung.

b. Vergleichen Sie die Worst-Case Komplexität (O-Notation) von Merge Sort mit der von Quick Sort und erklären Sie, falls erforderlich, die Unterschiede. Wann tritt gegebenenfalls der Worst-Case ein?

c. Nennen Sie jeweils ein Sortierverfahren bei dem bei einem bereits sortierten Feld minimaler bzw. maximaler Aufwand entsteht. Wie groß ist die Komplexität in diesen Fällen?

Aufgabe 6. (Heaps)

Gegeben sei das folgende Array, das einen Heap repräsentieren soll

35	18	14	4	20	9	7

a. Zeichnen Sie den zugehörigen Heap. Ist die Heap-Eigenschaft überall erfüllt? Korrigieren Sie gegebenenfalls die fehlerhaften Stellen durch Vertauschung möglichst weniger Elemente. Handelt es sich um einen Min-Heap oder einen Max-Heap?

b. In dem gegebenen Heap sollen die zwei größten Elemente gelesen (nicht gelöscht) werden. Beschreiben Sie den Lesevorgang und geben Sie dessen Komplexität in O-Notation an.

c. Beschreiben Sie den Vorgang des Einfügens eines neuen Elementes in einen Heap am Beispiel des oben gegebenen Heaps und des Schlüssels 21. Achten Sie darauf, dass die Heap-Eigenschaft auch nach dem Einfügen noch erfüllt sein muss. Geben Sie nach jeder Vertauschung von Elementen das zum Heap gehörende vollständige Array an.

Aufgabe 7. (Rekursion)

Erläutern Sie die Begriffe Abstieg und Aufstieg bei rekursiven Funktionen anhand des folgenden Beispielprogramms:

```
#include<stdio.h>

void functionRek (int n)
{
    if (n>1)
    {
        functionRek(n/2);
        printf("%d ", n);
    }
}

void main (void)
{
    functionRek(16);
}
```

Welche Ausgabe erzeugt dieses Programm?

Aufgabe 8. (Bäume)

Gegeben sei die folgende Liste von geordneten Paaren (x, y)

(dog, 20)
(pig, 18)
(cat, 4)
(mug, 32)
(man, 28)
(arm, 12)
(bit, 9)
(gag, 40)
(hex, 23)
(ice, 19)
(pin, 2)
(rug, 13)
(sly, 27)
(tub, 25)
(van, 15)

in der x den Schlüssel und y die Priorität enthält. Erzeugen Sie einen binären Baum, der diese Paare enthält, so dass

a. der Baum ein binärer Suchbaum bezüglich der Schlüssel ist. Dabei wird eine lexikographische Ordnung der Schlüssel angenommen. Welche Traversierungsmethode liefert die Knoten in alphabetischer Reihenfolge?

b. der Baum bezüglich der Priorität die Heapeigenschaft erfüllt.

Hinweis: Es reicht, wenn Sie bei Aufgabenteil a. nur die Schlüssel und bei b. nur die Prioritäten einzeichnen.

Aufgabe 9. (Suchbaum)

Die Liste aller Knoten eines **binären Suchbaumes** B, der in Preorder-Traversierung durchlaufen wird, sei gegeben als:

7 4 2 3 5 8 9

a) Welche Eigenschaften hat ein binärer Suchbaum?
b) Geben Sie den Suchbaum B zu der gegebenen Knotenfolge an.
c) Welche Knotenreihenfolge ergibt eine Postorder-Traversierung dieses Baumes B?
d) Handelt es sich bei diesem Baum B um einen Min-Heap? Begründen Sie Ihre Antwort.

Aufgabe 10. (Komplexität)

Welche Komplexität in O-Notation hat die folgende Funktion bezogen auf den Parameter `n`?

```
void funk (int n)
    int i = 0;
    while (i<=n) {
        printf("%d ", i);
        i +=4;
    }

    printf("\nn = %d\n", n);
    printf("fertig\n");
}
```

Aufgabe 11. (Binärbaum) → Klausuraufgabe SS 2008

Entwickeln Sie ein Unterprogramm

```
int anzVerkettungen (BiTreeNode *root);
```

welches in einem Binärbaum die ***Anzahl aller Verkettungszeiger, die ungleich NULL sind,*** zählt. Der Binärbaum liege dabei als Sammlung von Knoten vor, welche durch die Verkettungszeiger miteinander verbunden sind. Verkettungszeiger, die zu keinem Kind-Knoten führen, sind auf NULL gesetzt Die Knoten sind vom Typ

```
typedef struct BiTreeNode_ {
     void              *data;
     struct BiTreeNode_ *left;
     struct BiTreeNode_ *right;
} BiTreeNode;
```

Der Übergabeparameter der Funktion (root) zeige auf den Wurzelknoten des zu analysierenden Binärbaums und der Rückgabewert sei die ermittelte Verzweigungszahl. Falls

der Baum leer ist oder nur aus einem Knoten besteht, soll die Funktion den Wert 0 zurück liefern.