Aufgabenblatt zur 8. Übung

Zeitraum: 06.12. bis 10.12.2010

1. Aufgabe: Klausuraufgabe 07.2010 (AGS 3.41)

Gegeben seien die Typdefinitionen:

```
typedef struct elem *IntList;
typedef struct elem { int key;
                      IntList next;
                    } elemtype;
```

(a) Schreiben Sie in *C* eine Funktion `IntList head(IntList *l)`, die von einer Liste das Kopfelement liefert und dieses aus der Liste entfernt. Bei einer leeren Liste soll `NULL` geliefert werden.

(b) Schreiben Sie in *C* eine Funktion `Int aufsteigend(IntList l)`, die testet, ob die Liste `l` aufsteigend sortiert ist. Geben Sie eine iterative **und** eine rekursive Funktion an. (Hinweis: in der Programmiersprache `C` steht der Wert `0` für *falsch* und ein Wert ungleich `0` für *wahr*.)

2. Aufgabe: (AGS 3.22)

Die folgende Typdeklaration für Elemente eines Binärbaumes sei gegeben:

```
typedef struct node *B_Ptr;
typedef struct node {
  int key;
  int weight;
  B_Ptr left, right;
} bnodetype;

B_Ptr Tree;
```

(a) Geben Sie in *C* eine rekursive Funktion `void teilsumme(B_Ptr t)` an, die bei jedem Knoten eines beliebigen Binärbaumes `Tree` des oben aufgeführten Typs in `weight` die Summe aller Schlüsselwerte des Unterbaumes einträgt, der diesen Knoten als Wurzelknoten hat.

(b) Schreiben Sie in *C* eine rekursive Funktion `void gewicht(B_Ptr t)`, die bei jedem Knoten eines beliebigen Binärbaumes `Tree` des oben aufgeführten Typs in `weight` einträgt:
0, falls die Summen der Schlüsselwerte (key) beider Unterbäume gleich sind
1, falls die Summe der Schlüsselwerte im rechten Unterbaum größer ist als die im linken
-1, falls die Summe der Schlüsselwerte im rechten Unterbaum kleiner ist als die im linken

Hinweis: Schreiben Sie gegebenenfalls eine Hilfsfunktion zur Berechnung der Summe der Schlüsselwerte eines Baumes.

3. Aufgabe: (AGS 3.46)

Folgender Ausschnitt aus einem korrekten `main`–Programm in *C* sei gegeben:

```
#incluce <stdio.h>
#include "mymodul.h"

...

int main() {
  ...
  person a;
  int b, c[20];
  ...
  f1(c, &b);
  a = f2(b);
  ...
}
```

Es sollen `person`, `f1` und `f2` Objekte sein, die durch den Modul `mymodul` bereitgestellt werden. `person` soll dabei ein Strukturtyp sein, in dem Name und Vorname mit jeweils maximal 20 Zeichen deklariert werden. Geben Sie für diesen Sachverhalt die relevanten Definitionen in `mymodul.h` und `mymodul.c` an.

Gehen Sie davon aus, dass die Funktionsrümpfe bekannt sind. Deuten Sie diese jeweils durch `{...}` an.

Zusatzaufgabe 1: (AGS 3.25*)

Gegeben sei folgende Typdeklaration für bewertete binäre Bäume:

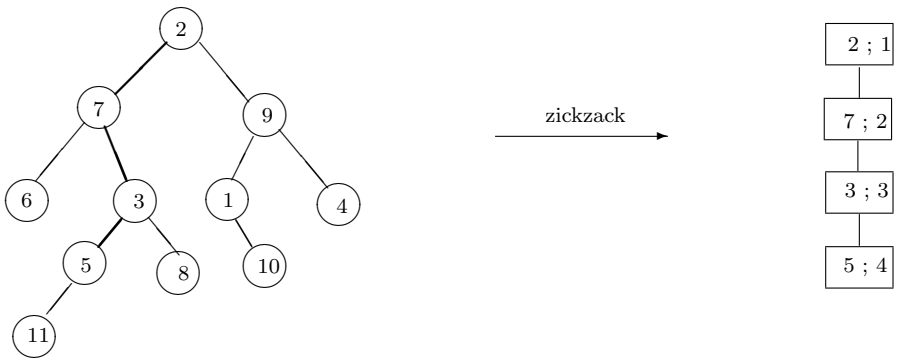
```
typedef struct node *BPtr;
typedef struct node {  int wert;
                         BPtr left,right;
                       } nodetype;
```

Aus einem bewerteten binären Baum t soll nun eine Liste l wie folgt erzeugt werden:

Der Baum t ist, von der Wurzel beginnend, auf einem links–rechts–Pfad zu durchlaufen.
Für jeden auf diesem Weg durchlaufenen Knoten soll
- ein Listenelement erzeugt werden,
- dieses an die bestehende Liste l angehängt werden
- und in das Listenelement die Bewertung des aktuell durchlaufenen Knotens sowie seine Pfadlänge gespeichert werden.

Der Pfad endet in dem Knoten, wo es keinen entsprechenden Nachfolgerknoten mehr gibt, spätestens also in einem Blatt.

Beispiel: bewerteter binärer Baum t Liste l



Hinweise:
links–rechts–Pfad bedeutet, daß der Pfad wechselweise zum linken bzw. rechten Nachfolgerknoten fortgesetzt wird. Der Pfad beginnt am Wurzelknoten des Baumes mit einem linken Nachfolgerknoten.
Pfadlänge zu einem Knoten n : Anzahl der Knoten des Pfades von der Wurzel zu n.
Die Wurzel hat die Pfadlänge 1.

(a) Geben Sie eine geeignete Datenstruktur für die gewünschte Liste l mit dem Bezeichner `list` in C an.

(b) Geben Sie eine `void`–Funktion `zickzack` in C an, die zu jedem bewerteten binären Baum t eine Liste wie oben beschrieben ausgibt. Der Funktionskopf muss mindestens zwei formale Parameter besitzen: nämlich einen Zeiger *baum* vom Typ `BPtr` und einen Zeiger *liste*, der bei Aufruf von `zickzack` auf den Kopf der zu erzeugenden Liste l zeigen soll.

(c) Geben Sie einen Funktionsaufruf von `zickzack` an. Definieren Sie ggf. die benutzten Variablen. Sie können dabei annehmen, dass der übergebene Zeiger vom Typ `BPtr` auf die Wurzel des Baumes zeigt.

Zusatzaufgabe 2: (AGS 3.45*)

Programmieren Sie einen Modul `komplZahlen`, der für die Arithmetik mit komplexen Zahlen die Funktionen `addiere`, `subtrahiere`, `multipliziere` und `dividiere` exportiert und damit der allgemeinen Nutzung zugänglich macht.

Verwenden Sie für die Darstellung der komplexen Zahlen den (noch anzugebenden) Typ:

```
struct Komplex {
  float re, im;
}
```

Die oben genannten Funktionen sollen den Ergebnistyp `komplexType` haben.

Testen Sie mit Hilfe eines Hauptprogrammes und entsprechenden Eingaben die Funktionstüchtigkeit Ihres Moduls `komplZahlen`.