DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart

Bitte überprüfen Sie vor Beginn der Prüfung

1. Sind Sie gesund und prüfungsfähig?
2. Sind Ihre Tasche und sämtliche Unterlagen seitlich an der Wand abgestellt und nicht am Klausurtisch?
3. Haben Sie auch außerhalb des Klausurraumes im Gebäude keine Unterlagen liegen lassen?

1. Semester: Wiederholungsprüfung: Teilklausur I in Informatik
Kurs TEL 09 GR 2

Matrikel-Nummer: **Name:**			
Bearbeitungszeit:	**90 Minuten**	**Studienfach:**	
Hilfsmittel:	**keine**	**Dozent:**	**Dr. S. Lewandowski**
		Datum:	**09.09.2010**

Aufgabe	max. Punkte	erreichte Punkte
1: Kleine und große Zahlen (Struktogramm)	12	
2: Datum auf Rigel 8 (DH, C/C++)	8	
3: Dauer auf Rigel 8 (DF, PA, C/C++)	34	
4: Anzahl der Elemente (Ringlisten, C/C++)	30	
Summe	80 (+4)	

Es sind insgesamt 84 Punkte zu erreichen (80 Punkte werden als 100% gewertet)
(Jeder Punkt entspricht einer Bearbeitungszeit von etwa 1 Minute)

Bitte kennzeichnen Sie die Aufgabenstellung und jedes zusätzliche Blatt lesbar mit Ihrem Namen und Ihrer Matrikelnummer!

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

Aufgabe 1 — 12 Punkte

(12 Punkte) **Kleine und große Zahlen:** Entwerfen Sie ein kurzes Programm, das von einem Benutzer Zahlen einliest und für diese die kleinste und größte Zahl bestimmt. Das Ende erfolgt durch eine Eingabe einer Zahl kleiner oder gleich 0 durch den Benutzer. Das Programm soll danach ausgeben, welches die kleinste und welches die größte vom Benutzer eingegebene Zahl war. Wurde die Eingabe sofort mit einer Zahl kleiner oder gleich 0 abgebrochen, so soll die Ausgabe lauten: `Es wurden keine Zahlen eingegeben.`

Beispiel: Der Benutzer gibt folgende Zahlen ein (jeweils getrennt durch die <RETURN>-Taste): `42 11 7 21 18 4 3 -2`, dann soll die Ausgabe lauten:
`Kleinste Zahl war die 3, größte Zahl war die 42.`

Geben Sie Ihren Entwurf als Struktogramm an (12 Punkte). Der Entwurf soll so detailliert sein, dass eine Umsetzung 1:1 in ein C/C++-Programm möglich wäre, d.h., dass alle Fallunterscheidungen und Schleifen auch im Entwurf als solche vorkommen müssen. Beachten Sie: Eine Umsetzung in C/C++ ist hier *nicht* gefordert.

Aufgabe 2 — 8 Punkte

(4+4 = 8 Punkte) In dieser Aufgabe ist eine **Datenstruktur** zu entwerfen, als Datenhierarchiediagramm darzustellen und in C/C++-Code umzusetzen. Die Datenstruktur soll zur Verwaltung einer Datumsangabe auf Rigel 8 dienen. Ein Datum auf Rigel 8 setzt sich zusammen aus einer Jahreszahl, einer Monatsangabe (möglich sind hier `A`, `B` und `C`) und dem Tag im Monat (jeder Monat hat 12 Tage).

Geben Sie ein Datenhierarchiediagramm für Ihre Datenstruktur an (4 Punkte) und geben Sie die Umsetzung in C/C++-Code an (4 Punkte).

Aufgabe 3 — 34 Punkte

(2+18+14 = 34 Punkte) **Datum auf Rigel 8:** In dieser Aufgabe ist eine Funktion `dauer(.,.)` zu entwerfen und in C- oder C++-Code umzusetzen.

Die Funktion soll dabei zwei Datumsangaben (aus Aufgabe 2) als Parameter bekommen und die Differenz der beiden Datumsangaben berechnen. Die Ausgabe soll dabei getrennt nach Jahren, Monaten und Tagen erfolgen. Ist ein Wert 0, so soll dieser in der Ausgabe übersprungen werden.

Beispiel 1: Ist das frühere Datum `(2010,A,9)` und das spätere Datum `(2010,C,9)` so soll die Ausgabe lauten: `Die Dauer beträgt 2 Monate.`

Beispiel 2: Ist das frühere Datum `(2008,B,12)` und das spätere Datum `(2010,B,2)` so soll die Ausgabe lauten: `Die Dauer beträgt 1 Jahr, 2 Monate, 2 Tage.`

Beispiel 3: Ist das frühere Datum `(2010,B,12)` und das spätere Datum `(2010,A,2)` so soll die Ausgabe lauten: `Das zweite Datum liegt vor dem ersten!`

Beispiel 4: Ist das frühere Datum `(2010,A,9)` und das spätere Datum `(2010,A,9)` so soll die Ausgabe lauten: `Beide Datumsangaben sind identisch.`

Beispiel 5: Ist das frühere Datum `(2010,F,0)` und das spätere Datum `(2010,F,0)` so soll die Ausgabe lauten: `Mindestens eine der beiden Datumsangaben ist ungültig!`

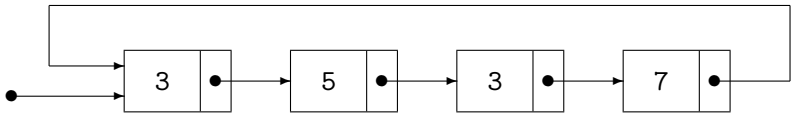
Geben Sie das Datenflussdiagramm (DF) der Ebene 0 an (Funktion als ein Kästchen) (2 Punkte), sowie einen Programmablaufplan für die von Ihnen entworfene Funktion mit den nötigen Verfeinerungen (18 Punkte). Der Entwurf soll so detailliert sein, dass eine Umsetzung 1:1 in ein C/C++-Programm möglich ist, d. h., dass alle Fallunterscheidungen und Schleifen auch im Entwurf als solche vorkommen müssen. Beachten Sie: die Parameter sind an die Funktion zu übergeben und werden von dieser zurückgeliefert, Sie müssen sich *nicht* um die Ein- und Ausgabe der Werte kümmern. Geben Sie schließlich noch die Umsetzung der Funktion(en) in C/C++ an (14 Punkte).

Hinweis: Falls Sie Aufgabe 2 nicht lösen konnten, verwenden Sie als Parameter für eine Datumsangabe entweder ein `int`-Array mit 3 Werten oder wahlweise drei einzelne `int`-Werte.

Aufgabe 4 — 30 Punkte

(4+26 = 30 Punkte) **Einfach verkettete Ringlisten:** In dieser Aufgabe beschäftigen wir uns mit einfach verketteten Ringlisten, der Anker zeige dabei auf ein beliebiges Element der Ringliste.

1. (4 Punkte) Geben Sie eine C- oder C++-Typdefinition für eine einfach verkettete Ringliste aus ganzen Zahlen an. Der Datentyp für den Anker soll dabei `RgLstPtr` heißen.
2. (26 Punkte) Schreiben Sie eine C/C++-Funktion `void anz(RgLstPtr liste)`, die als Parameter den Anker auf eine einfach verkettete Ringliste bekommt. Ist die Liste leer, so soll die Meldung „`Liste ist leer.`“ ausgegeben werden, ansonsten die Anzahl der verschiedenen Elemente in der Ringliste. Zu der Liste



soll die Ausgabe lauten: „`Die Ringliste enthaelt 3 verschiedene Element(e).`“

Beachten Sie: der Parameter ist an die Funktion zu übergeben, Sie müssen sich *nicht* um die Eingabe der Elemente kümmern. Ihre Funktion braucht nicht zu überprüfen, ob es sich wirklich um eine Ringliste handelt.

Hinweis 1: eine Möglichkeit der Lösung der Aufgabe ist, sich eine Hilfsfunktion zu schreiben, die überprüft, ob eine Zahl zwischen Anker und einem zweiten Element vorkommt. Überlegen Sie, wie Sie solch eine Hilfsfunktion zur Lösung der obigen Aufgabe nutzen können.

Hinweis 2: Sie können auch einen anderen Lösungsansatz verfolgen.

Hinweis 3: Wenn Sie die Aufgabe nicht lösen können, versuchen Sie die Anzahl der Elemente in der Ringliste zu bestimmen (ohne Berücksichtigung, ob Elemente gleich sind) (bis zu 14 Punkte).

DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart

Bitte überprüfen Sie vor Beginn der Prüfung

1. Sind Sie gesund und prüfungsfähig?
2. Sind Ihre Tasche und sämtliche Unterlagen seitlich an der Wand abgestellt und nicht am Klausurtisch?
3. Haben Sie auch außerhalb des Klausurraumes im Gebäude keine Unterlagen liegen lassen?

2. Semester: Wiederholungsprüfung: Teilklausur II in Informatik
Kurs TEL 09 GR 2

Matrikel-Nummer:			
Name:			
Bearbeitungszeit:	**90 Minuten**	**Studienfach:**	
Hilfsmittel:	**keine**	**Dozent:**	**Dr. S. Lewandowski**
		Datum:	**09.09.2010**

Aufgabe	max. Punkte	erreichte Punkte
1: Binäre Suche, Interpolationssuche (durchführen)	6	
2: Löschen in bin. Suchbäumen (beschr., durchführen)	8	
3: Einfügen in AVL-Bäumen (durchführen)	12	
4: Insertionsort (durchführen, Fehlstand)	11	
5: Quicksort (durchführen)	10	
6: Dijkstra-Algorithmus (durchführen)	12	
7: Ein Suchbaumalgorithmus (C/C++, O-Notation)	25	
Summe	80 (+4)	

Teilkl. Info. I:	Teilkl. Info. II:	Σ	Note:

Es sind insgesamt 84 Punkte zu erreichen (80 Punkte werden als 100% gewertet)
(Jeder Punkt entspricht einer Bearbeitungszeit von etwa 1 Minute)

Bearbeiten Sie die Aufgaben bitte direkt auf der Aufgabenstellung! Bitte kennzeichnen Sie die Aufgabenstellung und jedes zusätzliche Blatt lesbar mit Ihrem Namen und Ihrer Matrikelnummer!

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

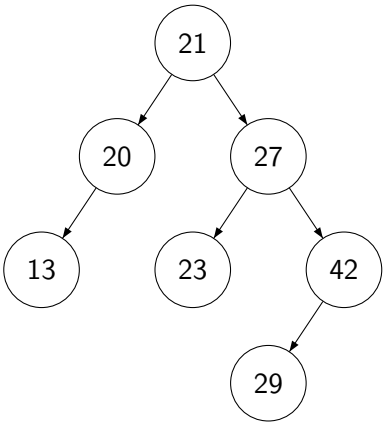
Aufgabe 1 — 6 Punkte

(3+3 = 6 Punkte) **Binäre Suche, Interpolationssuche:** Gegeben ist ein Array mit den Werten (13,14,17,18,33,36,37). Es soll der Wert 17 gesucht werden.

- (3 Punkte) Geben Sie an, mit welchen Elementen bei **binärer Suche** die 17 verglichen wird, bis sie gefunden wird.
- (3 Punkte) Geben Sie an, mit welchen Elementen bei **Interpolationssuche** die 17 verglichen wird, bis sie gefunden wird.

Aufgabe 2 — 8 Punkte

(5+3 = 8 Punkte) **Löschen in binären Suchbäumen:** Gegeben ist folgender binäre Suchbaum:



- (5 Punkte) Werden in einem binären Suchbaum Knoten mit zwei Kindern gelöscht, so werden diese durch das nächstgrößere (Inordernachfolger) oder nächstkleinere Element (Inordervorgänger) ersetzt. Geben Sie die beiden Suchbäume an, die durch Löschen des Elements 21 entstehen können.

  Was muss man nach dem Ersetzen des zu löschenden Elements noch tun?
- (3 Punkte) Beschreiben Sie in 2 Sätzen, wie der Inordervorgänger oder Inordernachfolger gefunden werden kann.

Aufgabe 3 — 12 Punkte

(12 Punkte) Fügen Sie in einen anfangs leeren **AVL-Baum** folgende Zahlen in dieser Reihenfolge ein: 2, 4, 8, 5, 7, 9, 6. Wenn eine Rotation notwendig ist, geben Sie den AVL-Baum jeweils vor und nach dieser Rotation an. Geben Sie in jedem Fall den AVL-Baum an, nachdem alle Elemente eingefügt wurden.

Aufgabe 4 — 11 Punkte

(9+2 Punkte) **Insertionsort:** Gegeben ist ein Array mit den Zahlen (8,6,2,3,1,5,4).

(9 Punkte) Führen Sie Insertionsort durch:

- Geben Sie den Inhalt des Arrays nach jedem kompletten Durchlauf der äußeren Schleife an.
- Tragen Sie jeweils die Anzahl der in diesem Durchlauf durchgeführten Vertauschungen ein.

Hinweis: die Anzahl der Zeilen in der Tabelle ist größer als die tatsächlich auszufüllende Anzahl der Zeilen.

Durchlauf								Vertauschungen
vor 1.	8	6	2	3	1	5	4	-
nach 1.								
nach 2.								
nach 3.								
nach 4.								
nach 5.								
nach 6.								
nach 7.								
nach 8.								
nach 9.								
nach 10.								
nach 11.								
nach 12.								
nach 13.								
nach 14.								
nach 15.								

(2 Punkte) Wie groß ist der Fehlstand (Inversionszahl) des obigen Feldes (8,6,2,3,1,5,4)?

Aufgabe 5 — 10 Punkte

(10 Punkte) Bei **Quicksort** wird als Pivot-Element das Element in der Mitte gewählt (Index `(l+r)/2`). Folgendes Feld sei gegeben. Führen Sie die erste Iteration bei Quicksort durch (also Wandern der Zeiger mit Vertauschungen bis die beiden Zeiger übereinander hinweg gelaufen sind) – in Ihrer Bearbeitung muss ersichtlich sein, welche Elemente getauscht wurden und nach welchem Kriterium das Wandern der Zeiger beendet wird:

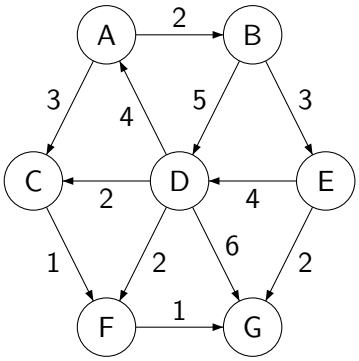
11 73 16 15 67 7 46 50 32 57 92 76 42 89 95 56 21 20 74

Geben Sie an, mit welchen Indexgrenzen die beiden resultierenden rekursiven Aufrufe von Quicksort erfolgen.

Aufgabe 6 — 12 Punkte

(9+3 Punkte) **Kürzeste-Wege-Berechnung mit dem Algorithmus von Dijkstra:**

Führen Sie den Dijkstra-Algorithmus auf folgendem Graphen aus:
Zu Beginn ist lediglich die Entfernung zum Startknoten A mit d(A)=0 bekannt. Im Dijkstra-Algorithmus wird in jeder Iteration ein Knoten aus der Randmenge R entfernt. (9 Punkte) Geben Sie in jeder Zeile den Zustand der Menge R jeweils vor diesem Entfernen eines Knotens an, den durch delete_min gewählten Knoten und die berechneten Entfernungen d() am Ende des Durchlaufs für alle Knoten.



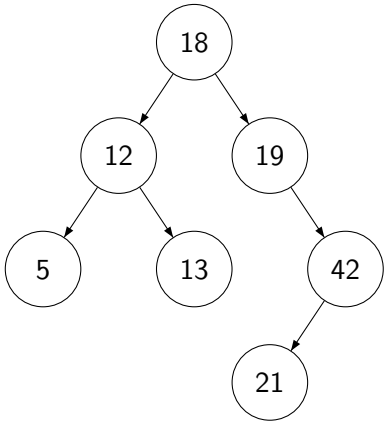
	Menge R	Ergebnis von delete_min(R)	berechnete Entfernungen A	B	C	D	E	F	G
0.	$\emptyset$	„Initialisierung“	0	∞	∞	∞	∞	∞	∞
1.	{A}								
2.									
3.									
4.									
5.									
6.									
7.									
8.									

(3 Punkte) Geben Sie den Kürzeste-Wege-Baum an, also den Baum, der durch die während der Berechnung gemerkten Vorgänger (`pred[]`) bestimmt ist.

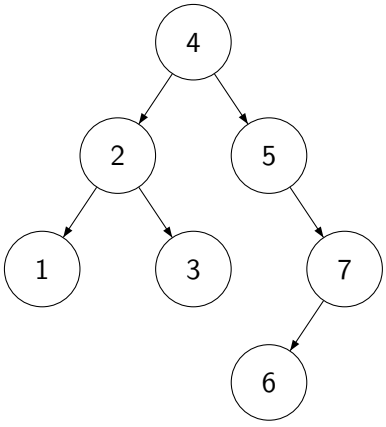
Aufgabe 7 — 25 Punkte

(2+8+10+2+3 = 25 Punkte) **Ein Suchbaumalgorithmus:** In dieser Aufgabe soll zunächst zu einem gegebenen binären Suchbaum ein strukturgleicher Binärbaum erstellt werden, sodass jede Knotenbeschriftung der Bearbeitungsnummer in der Inorderreihenfolge entspricht. Mithilfe dieses so erstellten Baumes kann nun das k-kleinste Element im Suchbaum schnell ermittelt werden.

Ein Beispiel: zu dem Baum aus Aufgabe 2



wird zunächst der folgende Baum erstellt



Soll nun das 3-te Element im linken Suchbaum gefunden werden, so ist die Wurzel der 4-te Knoten, das gesuchte Element ist somit im linken Unterbaum zu finden. Der Wurzel im linken Unterbaum, der 12, ist die Inordernummer 2 zugeordnet, wir suchen in dssen rechtem Unterbaum weiter. Dessen Wurzel hat die Inordernummer 3, ist somit das gesuchte Element: die 13.

- (2 Punkte) Geben Sie eine C/C++-Datenstruktur für einen binären Suchbaum oder Binärbaum an. Der Datentyp für eine Variable, die auf die Wurzel eines solchen Baums zeigt, soll dabei `bbPtr` heißen.
- (8 Punkte) Schreiben Sie eine C/C++-Funktion `bbPtr inorderKopie(const bbPtr root)`, die als Parameter den Zeiger auf die Wurzel des binären Suchbaums bekommt, der kopiert werden soll, und als Ergebnis den Zeiger auf die Wurzel des erstellten Baumes zurückliefert. (Wenn Sie diese Aufgabe nicht komplett lösen können, schreiben Sie alternativ eine C/C++-Funktion, die in einem Binärbaum lediglich die Knotenbeschriftung so ändert, dass diese jeweils der Nummer im Inorderdurchlauf entspricht (bis zu 5 Punkte).)
- (10 Punkte) Schreiben Sie eine weitere C/C++-Funktion `int baumselect(const bbPtr root, const bbPtr ioKopie, const int k)`, die als Parameter den Zeiger auf die Wurzel des binären Suchbaums bekommt, in dem gesucht werden soll, einen weiteren Zeiger auf die Wurzel des Binärbaums, der durch die obige Funktion `inorderKopie` entstanden ist, sowie eine Zahl `k`. Rückgabewert der Funktion soll das nach oben beschriebener Methode gefundene Element sein (falls `k` zu klein oder zu groß ist, so sei das Ergebnis `-1`).
- (2 Punkte) Geben Sie den Aufwand Ihres Algorithmus `inorderKopie(...)` in O-Notation an.
- (3 Punkte) Mit welchem Aufwand im Mittel würden Sie für einen Aufruf Ihrer Funktion `baumselect(...)` rechnen?

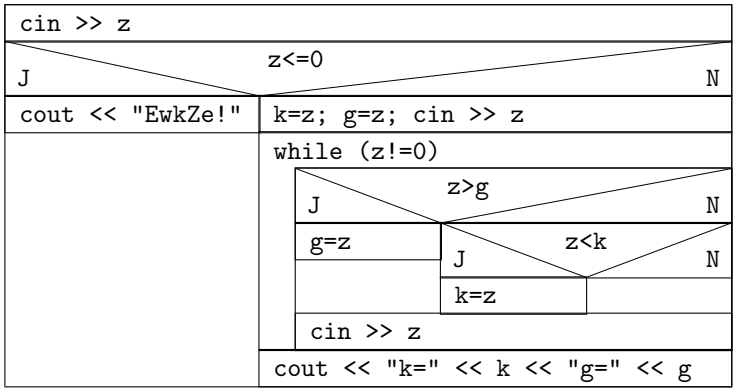
Beachten Sie: die Parameter sind an die Funktion zu übergeben, Sie müssen sich *nicht* um die Eingabe eines Baumes kümmern.

Zusatzaufgabe: Mit welchem Aufwand würden Sie rechnen, wenn im linken Suchbaum ein neuer Knoten eingefügt wird und der der rechte Baum angepasst werden muss (beachten Sie, dieser muss nicht komplett neu erstellt werden).

Teilklausur I (Wdh.) in Informatik – Lösungshinweise

Aufgabe 1 — Kleine und große Zahlen

Eine Schleife, in der sich abhängig von einer mehrfachen Fallunterscheidung zwei Werte gemerkt werden. Wichtig ist hier die Initialisierung der beiden Werte (sinnvoll z. B. mit der ersten eingelesenen Zahl). Ebenso lässt sich der Fall, dass keine Zahlen eingelesen wurden, einfach und kompliziert umsetzen (hier einfacher ohne Zählen der Werte).



Bei der Bearbeitung ist unerheblich, ob die Befehle (wie hier) C/C++-Syntax haben oder Pseudo-Code sind.

Aufgabe 2 — Datum auf Rigel 8

Wir geben hier nur den C++-Code an:

```
struct Datum {
  int jahr;
  char monat;
  int tag;
};
```

Aufgabe 3 — Dauer auf Rigel 8

Es bietet sich an, jeweils die Tage seit Beginn der Zeitrechnung zu bestimmen. Wir geben hier nur den C++-Code an:

```
void istGueltig(const Datum & d, bool & b) {
  b = ((d.jahr>=0) && (d.monat=='A' || d.monat=='B' || d.monat=='C') && (d.tag>=1 && d.tag<=12));
}

void tageSeitNull(const Datum & d, int & a) {
  int m;  if (d.monat=='A') m=1; else if (d.monat=='B') m=2; else m=3;
  a=(d.jahr-1)*36+(m-1)*12+d.tag;
}
```

```
void dauer(const Datum & d1, const Datum & d2) {
  int a1, a2, j, m, t;
  bool b1, b2; istGueltig(d1,b1); istGueltig(d2,b2);
  if(b1 && b2) {
    a1=tageSeitNull(d1); a2=tageSeitNull(d2);
    if(a2==a1) cout << "Beide Datumsangaben sind identisch.";
    else if (a2>a1) {
      j=(a2-a1)/36; m=((a2-a1)%36)/3; t=(a2-a1)%12;
      cout << "Die Dauer beträgt ";
      if (j!=0) if (j==1) cout << "1 Jahr"; else cout << j << " Jahre";
      if (m==0 || t==0) cout << '.';
      else {
        cout << ',';
        if (m!=0) {
          if (m==1) cout << "1 Monat"; else cout << m << " Monate";
          if (t!=0) cout << ','; else cout << '.';
        }
        if (t==1) cout << "1 Tag."; else if (t>=1) cout << t << " Tage.";
      }
    } else cout << "Das zweite Datum liegt vor dem ersten!";
  } else cout << "Mindestens eine der beiden Datumsangaben ist ungültig!";
}
```

Ohne Berücksichtigung der Komma-oder-Punkt-Problematik und der Einzahl-Mehrzahl-Problematik vereinfachen sich die Fallunterscheidungen erheblich.

Aufgabe 4 — Ringlisten

Solange noch nicht jedes Element in der Ringliste betrachtet wurde, prüfe für dieses jeweils, ob es davor schonmal vorkam. Wir geben für diese Idee hier den C++-Code an:

```
struct RgLst {
  int val;
  RgLst *next;
};
typedef RgLst * RgLstPtr;

void anz(const RgLstPtr liste) {
  if (liste==NULL) cout << "Die Liste ist leer!";
  else {
    int anz=1; // das erste zählt immer
    RgLstPtr lauf, lbn=liste; // Last Before Next
    while (lbn->next!=liste) { // für alle vorherigen Elemente
      lauf=liste;
      bool drin=false;
      while (lauf!=lbn->next) {
        if (lauf->val == lbn->next->val) { drin=true; break; }
        lauf=lauf->next;
      }
      if (!drin) anz++;
    }
  }
  cout << "Die Ringliste enthält " << anz << " verschiedene Element(e).";
}
```

Teilklausur II (Wdh.) in Informatik – Lösungshinweise

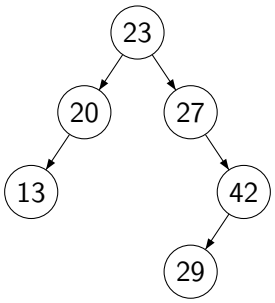
Aufgabe 1 — Binäre Suche, Interpolationssuche

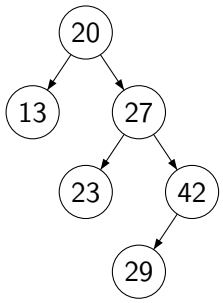
Gegeben war ein Array mit den Werten (13,14,17,18,33,36,37). Gesucht: der Wert $s = 17$.

- Bei **binärer Suche** wird jeweils in der Mitte des verbleibenden Teilfeldes gesucht ⇒ Vergleiche mit 18, 14, 17 ⇒ gefunden
- Bei **Interpolationssuche** wird jeweils am Index $m = l + (s - a[l]) \cdot (r - l)/(a[r] - a[l])$ gesucht ⇒ Vergleiche mit $a[0 + (17 - 13) \cdot (6 - 0)/(37 - 13)] = a[1] = 14$ und $a[2 + (17 - 17) \cdot (6 - 2)/(37 - 17) = a[2] = 17$ ⇒ gefunden

Aufgabe 2 — Löschen in binären Suchbäumen

Links der entstehende Suchbaum bei Ersetzen der 21 durch den Inordernachfolger, rechts bei Verwendung des Inordervorgängers. Die entstehende Lücke ist entweder ein Blatt (kann gelöscht werden) oder hat genau ein Kind (wird samt Unterbaum an die Lücke hochgezogen) – je einer der Fälle tritt hier auf.

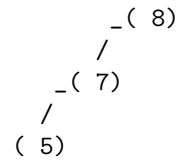




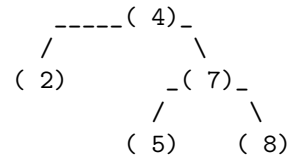
Aufgabe 3 — Einfügen in AVL-Bäume

```
Werte: 2, 4, 8                    ( 2)_
Balance an Knoten 2 verletzt           \
Unterbaum vor der Rotation:          ( 4)_
                                          \
                                        ( 8)

Linksrotation                       _( 4)_
Baum nach der Rotation:            /      \
                                 ( 2)     ( 8)


Nächste Werte: 5, 7                 _____( 8)
Balance an Knoten 8 verletzt       /
Unterbaum vor der Rotation:      ( 5)_
                                     \
                                   ( 7)
```

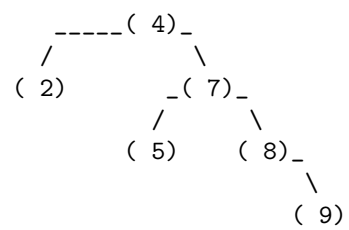
Linksrechtsrotation
Unterbaum nach der Linksrotation:



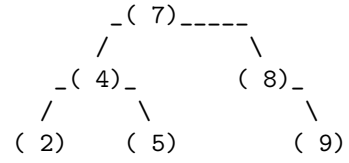
Baum nach der Rechtsrotation:



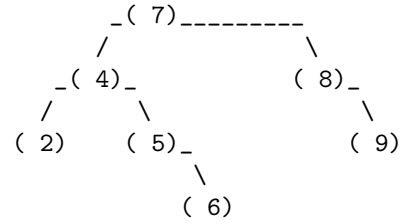
Nächster Wert: 9
Balance an Knoten 4 verletzt
Unterbaum vor der Rotation:



Linksrotation
Baum nach der Rotation:



Nächster Wert: 6



Aufgabe 4 — Insertionsort

Durchlauf								Vertauschungen
vor 1.	8	6	2	3	1	5	4	-
nach 1.	6	8	2	3	1	5	4	1
nach 2.	2	6	8	3	1	5	4	2
nach 3.	2	3	6	8	1	5	4	2
nach 4.	1	2	3	6	8	5	4	4
nach 5.	1	2	3	5	6	8	4	2
nach 6.	1	2	3	4	5	6	8	3

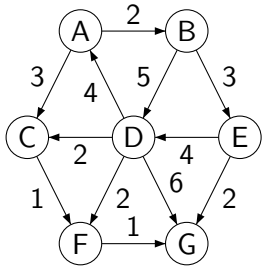
Der Fehlstand/Inversionszahl kann hier leicht als Summe der Vertauschungen ermittelt werden (zur Erinnerung: jeder Tausch eines benachbarten Pärchens verringert den Fehlstand um 1 – oder man zählt noch mal extra): das Ergebnis ist 14.

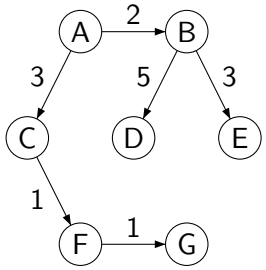
Aufgabe 5 — Quicksort

11 73 16 15 67 7 46 50 32 57 92 76 42 89 95 56 21 20 74

Pivot-Element ist die 57. Es werden zunächst die 73 und 20 vertauscht, dann die 67 und 21. Der linke Zeiger wandert dann bis zum Pivot-Element, der rechte zur 56 (Tausch 57 und 56). Ein weiterer Tausch der 92 mit 42, beide Zeiger stehen danach auf der 76. Die Schleife wird ein weiteres Mal betreten, der rechte Zeiger wandert noch ein Element nach links, es findet aber kein Tausch mehr statt, da die Zeiger bereits übereinander hinweggewandert sind. Die rekursiven Aufrufe haben die Parameter (0,10) und (11,18).

Aufgabe 6 — Dijkstra-Algorithmus





	Menge R	Ergebnis von delete_min(R)	berechnete Entfernungen A	B	C	D	E	F	G
0.	∅	„Initialisierung“	0	∞	∞	∞	∞	∞	∞
1.	{A}	A	0	2	3	∞	∞	∞	∞
2.	{B,C}	B	0	2	3	7	5	∞	∞
3.	{C,D,E}	C	0	2	3	7	5	4	∞
4.	{D,E,F}	F	0	2	3	7	5	4	5
5.	{D,E,G}	E	0	2	3	7	5	4	5
6.	{D,G}	G	0	2	3	7	5	4	5
7.	{D}	D	0	2	3	7	5	4	5
8.	∅	„fertig“	0	2	3	7	5	4	5

Aufgabe 7 — Ein Suchbaumalgorithmus

Die Datenstruktur:

```
struct bb
{
  int value;
  bb *left, *right;
};
typedef bb *bbPtr;
```

Die Inordernummerierung sollte durch einen separaten Baumdurchlauf bestimmt werden (identisch mit der Lösung der eingeschränkten Aufgabe):

```
void ionr(const bbPtr root, int & io)
{
  if(root!=NULL)
  {
    ionr(root->left,io);
    root->value=io++;
```

```
    ionr(root->right,io);
  }
}
```

Diese Funktion wird dann auch weiter verwendet, zusätzlich lagern wir die eigentliche Kopie aus, um die Inordernummerierung nicht rekursiv in jedem Kopierschritt anzustoßen.

```
bbPtr getKopie(const bbPtr root)
{
  if(root==NULL) return NULL;
  else
  {
    bbPtr node = new bb;
    node->left = getKopie(root->left);
    node->right = getKopie(root->right);
    return node;
  }
}

bbPtr inorderKopie(const bbPtr root)
{
  bbPtr ko=getKopie(root);
  if (ko!=NULL) { int io=1; ionr(ko,io); }
  return ko;
}
```

Der Aufwand für `inorderKopie(...)` ist – wie bei vielen Baumalgorithmen – linear (jeder Knoten wird einmal mit konstantem Aufwand betrachtet), also $O(n)$.

Bei der Selektionsaufgabe muss man sich in beiden Bäumen parallel bewegen:

```
int baumselect(const bbPtr root, const bbPtr ko, const int k)
{
  if(ko==NULL) return -1;
  else
  {
    if(k<ko->value) return baumselect(root->left,ko->left,k);
    else if(k==ko->value) return root->value;
    else return baumselect(root->right,ko->right,k);
  }
}
```

Der Aufwand im Mittel entspricht dem mittleren Aufwand zum Suchen in einem binären Suchbaum, ist also logarithmisch (genauer: es werden im Mittel $1.386 \cdot \log(n)$ Ebenen des Baumes betrachtet).

Hinweis zur Zusatzaufgabe: Wenn der Suchbaum geändert wird, so lässt sich zwar auf demgleichen Pfad die Kopie anpassen, im Mittel bekommen aber die Hälfte der Knoten neue Inordernummern!