# Wiederholungsklausur in Informatik I – TEL08GR1 + TELK08 (TEL08GR2) – Lewandowski

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

Aufgabe 1: Kleine und große Zahlen (14 Punkte) – Struktogramm
Aufgabe 2: Polynome (36 Punkte) – DF, PAD/PN, C/C++
Aufgabe 3: Eine Datenstruktur (10 Punkte) – DH, C/C++
Aufgabe 4: Einfach verkettete Listen (24 Punkte) – C/C++

Bitte kennzeichnen Sie jedes Blatt lesbar mit Ihrem Namen und Ihrer Matrikelnummer

Zugelassene Hilfsmittel: Keine

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

## Aufgabe 1 — 14 Punkte

(14 Punkte) **Kleine und große Zahlen:** Entwerfen Sie ein kleines Programm, das von einem Benutzer Zahlen einliest und sich dabei merkt, welches die kleinste und welches die größte der eingelesenen Zahlen war. Das Ende erfolgt durch eine Eingabe der Zahl 0 durch den Benutzer. Das Programm soll danach die kleinste und größte eingegebene Zahl ausgeben.

Beispiel: Der Benutzer gibt folgende Zahlen ein (jeweils getrennt durch die `<RETURN>`-Taste): `42 11 7 21 18 4 3 -2 0`, dann soll die Ausgabe lauten:
`Die kleinste Zahl war die -2, die größte Zahl war die 42.`

Beispiel 2: Der Benutzer gibt folgende Zahlen ein (jeweils getrennt durch die `<RETURN>`-Taste): `-21 0`, dann soll die Ausgabe lauten:
`Die kleinste Zahl war die -21, die größte Zahl war die -21.`

Beachten Sie, dass die 0 bei der Suche nach der kleinsten und größten Zahl nicht berücksichtigt wird. Bei sofortiger Eingabe der 0 soll die Ausgabe „`Es wurden keine Zahlen eingegeben!`“ lauten.

Geben Sie Ihren Entwurf als Struktogramm an (14 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 — 36 Punkte

(2+20+14 = 36 Punkte) **Polynome:** In dieser Aufgabe ist eine Funktion `printpoly(.,.)` zu entwerfen und in C- oder C++-Code umzusetzen.

Die Funktion soll dabei ein Array und dessen Größe als Parameter bekommen: Das Array enthält `int`-Werte, die die Koeffizienten eines Polynoms darstellen, wobei an Index $i$ der Koeffizient für $x^i$ steht. Die Funktion soll das Polynom auf dem Bildschirm ausgeben. Beispiele: zu dem Array `(4,1,0,-1,-2,3)` soll die Ausgabe „`f(x)=+3*x^5-2*x^4-x^3+x+4`“ lauten, zu dem Array `(0,0,2,0,-1)` soll die Ausgabe „`f(x)=-x^4+2*x^2`“ lauten.

Bei der Ausgabe sind folgende Regeln zu beachten (wie bei obigen Beispielen):

- ist der Koeffizient `1` oder `-1`, so soll der Faktor nicht angegeben werden, sondern direkt `+x^...` bzw. `-x^...`
- statt `x^1` ist nur `x` zu schreiben
- bei Koeffizient 0 wird der entsprechende Term weggelassen
- sind alle Koeffizienten gleich 0, so soll die Ausgabe „`f(x)=0`“ lauten

Hinweis: beachten Sie die Vorzeichen der Koeffizienten. Wie in obigem Beispiel ist für den ersten auszugebenden Koeffizienten ein führendes Pluszeichen erlaubt.

Geben Sie das Datenflussdiagramm (DF) der Ebene 0 an (Funktion als ein Kästchen) (2 Punkte), sowie einen Programmablaufplan mit Daten (PAD) oder ein Programmnetz (PN) für die von Ihnen entworfene Funktion mit den nötigen Verfeinerungen (20 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 Eingabe der Werte kümmern. Geben Sie schließlich noch die Umsetzung der Funktion in C/C++ an (14 Punkte).

## Aufgabe 3 — 10 Punkte

(5+5 = 10 Punkte) In dieser Aufgabe ist eine **Datenstruktur** zu entwerfen, als Datenhierarchiediagramm darzustellen und in C/C++-Code umzusetzen. Die Datenstruktur soll zur Verwaltung der Bücher einer Bibliothek dienen und soll dabei die Daten für ein Buch beinhalten. Folgende Daten sollen gespeichert werden: Vor- und Nachname des Autors, Titel, Erscheinungsjahr, sowie das Datum, wann das Buch für die Bibliothek gekauft wurde (speichern Sie für das Datum jeweils Zahlen für den Tag, Monat und das Jahr). Das Datum soll dabei ein eigener Datentyp sein.

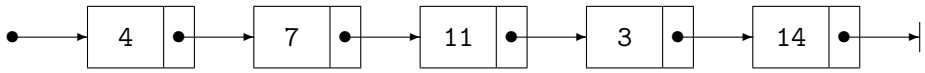
Geben Sie ein Datenhierarchiediagramm für Ihre Datenstruktur an (5 Punkte) und geben Sie die Umsetzung in C/C++-Code an (5 Punkte).

## Aufgabe 4 — 24 Punkte

(4+20 = 24 Punkte) **Einfach verkettete Listen:** In dieser Aufgabe beschäftigen wir uns mit einfach verketteten Listen, der Anker zeige dabei auf das erste Element.

1. (4 Punkte) Geben Sie eine C- oder C++-Typdefinition für eine einfach verkettete Liste aus `int`-Zahlen an. Der Datentyp für den Anker soll dabei `myList` heißen.

2. (20 Punkte) Schreiben Sie eine C/C++-Funktion `void dreierTest(myList liste)`, die als Parameter den Anker auf eine einfach verkettete Liste bekommt. Hat die Liste weniger als 3 Elemente, so soll die Meldung „`Liste ist zu kurz.`“ ausgegeben werden, ansonsten soll ermittelt werden, ob für jeweils drei direkt aufeinander folgende Elemente der Liste die Summe der ersten beiden Zahlen gleich der dritten Zahl ist, und wenn ja, diese Zahlen ausgegeben werden. Zu der Liste




soll die Ausgabe lauten: „`Die Summe von 4 und 7 ist 11. Die Summe von 11 und 3 ist 14.`“

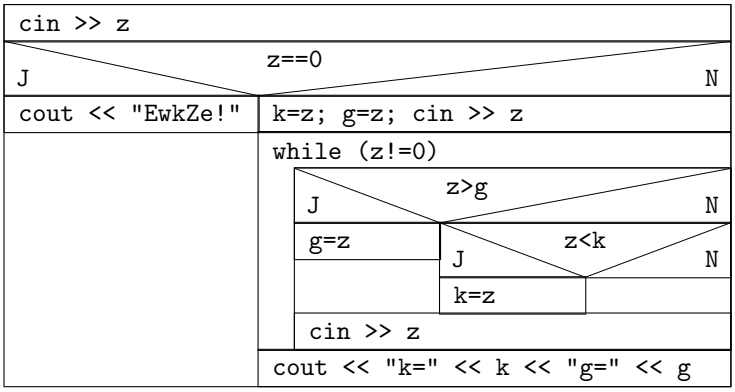
Beachten Sie: die Parameter sind an die Funktion zu übergeben, Sie müssen sich *nicht* um die Eingabe der Werte kümmern.

Beachten Sie weiterhin: die verschiedenen 3-Tupel können sich – wie oben im Beispiel – überlappen.

# Lösungshinweise – Wiederholungsklausur in Informatik I – Lewandowski

## Aufgabe 1 — Kleine und große Zahlen

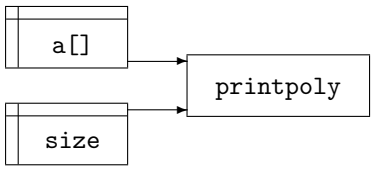
Eine Schleife, in der sich abhängig von einer mehrfachen Fallunterscheidung zwei Werte gemerkt werden. Wichtig ist hier die Initialisierung der beiden Werte (sinnvoll z.B. mit der ersten eingelesenen Zahl). Ebenso lässt sich der Fall, dass keine Zahlen eingelesen wurden, einfach und kompliziert umsetzen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




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

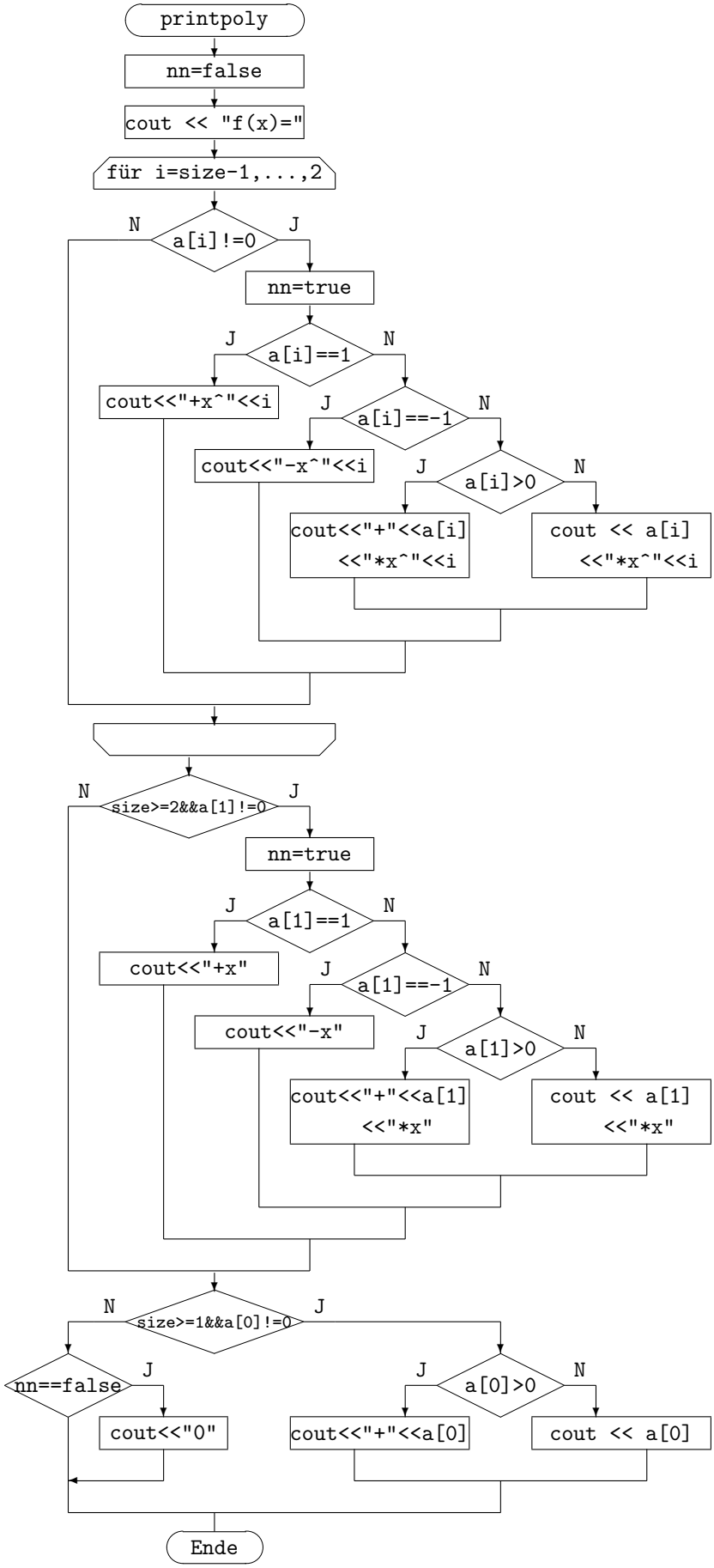
## Aufgabe 2 — Polynome

Das Datenflussdiagramm:




Am einfachsten war hier, die Fälle Konstante und linearer Term aus der Schleife für die Polynome höheren Grades herauszunehmen. Die Fallunterscheidungen sind jeweils sehr ähnlich und unterscheiden sich nur im Detail in der Ausgabe. Um am Ende entscheiden zu können, ob `f(x)=0` auszugeben ist, muss protokolliert werden (Variable `nn` (nicht null)), ob vorher wenigstens ein Term eines `x^k` ausgegeben wurde.

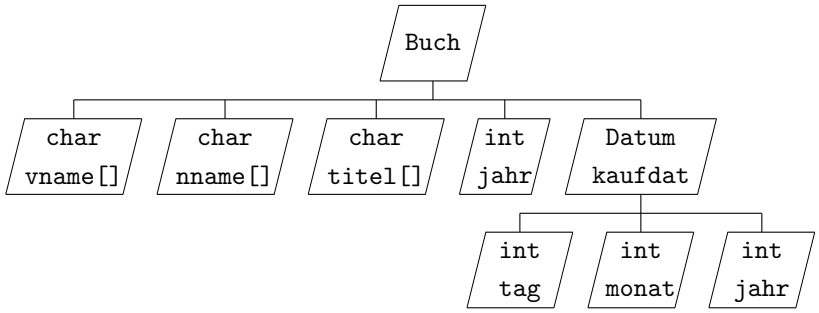
In der Abbildung des PAD wurde hier auf die Datenkästchen verzichtet - an jeden Befehl gehören links je ein Datenkästchen für die Variablen mit Lesezugriff (Pfeil zum Anweisungskästchen) und rechts je ein Datenkästchen für die Variablen mit Schreibzugriff (Pfeil zum Datenkästchen). Beim PN gäbe es für jede Variable im gesamten Diagramm nur ein Kästchen. Bei gleichzeitigem Lese- und Schreibzugriff wird der Pfeil in beide Richtungen gezeichnet.


printpoly
nn=false
cout << "f(x)="
für i=size-1,...,2
a[i]!=0
N
J
nn=true
a[i]==1
cout<<"+x^"<<i
a[i]==-1
cout<<"-x^"<<i
a[i]>0
cout<<"+"<<a[i]
<<"*x^"<<i
cout << a[i]
<<"*x^"<<i
size>=2&&a[1]!=0
nn=true
a[1]==1
cout<<"+x"
a[1]==-1
cout<<"-x"
a[1]>0
cout<<"+"<<a[1]
<<"*x"
cout << a[1]
<<"*x"
size>=1&&a[0]!=0
nn==false
cout<<"0"
a[0]>0
cout<<"+"<<a[0]
cout << a[0]
Ende

Die Umsetzung in C/C++:

```
void printpoly(const int * const a, const int size) {
  bool nn=false;
  cout << "f(x)=";
  for(int i=size-1;i>=2;i--) {
    if (a[i]!=0) {
      nn=true;
      if (a[i]==1) {
        cout << "+x^" << i;
      } else if (a[i]==-1) {
        cout << "-x^" << i;
      } else if (a[i]>1) {
        cout << "+" << a[i] << "*x^" << i;
      } else {
        cout << a[i] << "*x^" << i;
      }
    }
  }
  if (size>=2 && a[1]!=0) { // Arraygröße 2 sicherstellen
    nn=true;
    if (a[1]==1) {
      cout << "+x";
    } else if (a[1]==-1) {
      cout << "-x";
    } else if (a[1]>1) {
      cout << "+" << a[1] << "*x";
    } else {
      cout << a[1] << "*x";
    }
  }
  if (size>=1 && a[0]!=0) { // Arraygröße 1 sicherstellen
    if (a[0]>0) {
      cout << "+" << a[0];
    } else {
      cout << a[0];
    }
  } else if (nn==false) {
    cout << "0";
  }
}
```

## Aufgabe 3 — Eine Datenstruktur




```
struct Datum {
  int tag;
  int monat;
  int jahr;
};
struct Buch {
  char vname[20];
  char nname[30];
  char titel[40];
  int jahr;
  Datum kaufdat;
};
```

Bei der Umsetzung in C/C++ ist das Vorgehen von unten nach oben zu beachten:

## Aufgabe 4 — Einfach verkettete Listen

Man merke sich in drei Variablen e, z und d den ersten, zweiten und dritten Wert, vergleiche e+z==d (ggf. mit entsprechender Ausgabe). Der vorherige zweite Wert ist für das nächste Dreiertupel der erste Wert, der vorherige dritte nun der zweite. So ist es lediglich ein Durchlauf durch die Liste mit jeweiliger Aktualisierung der Werte e, z und d sowie eine Fallunterscheidung mit Ausgabe.

```
struct mayList {
  int value;
  mayList* next;
};
typedef mayList* myList;

void dreierTest(myList liste) {
  int e,z,d;
  if (liste!=NULL) {
    e=liste->value;
    liste=liste->next;
    if (liste!=NULL) {
      z=liste->value;
      liste=liste->next;
    }
  }
  if (liste==NULL) {
    cout << "Liste ist zu kurz.";
  } else {
    while (liste!=NULL) {
      d=liste->value;
      if (e+z==d) {
        cout << "Die Summe von " << e << " und " << z << " ist " << d << ". ";
      }
      e=z; z=d;
      liste=liste->next;
    }
  }
  cout << endl;
}
```

# Wiederholungsklausur in Informatik II – TEL08GR1 + TELK08 (TEL08GR2) – Lewandowski

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

Aufgabe 1: Binäre Suche, Interpolationssuche (7 Punkte) – durchführen (Verständnisfrage)
Aufgabe 2: Baumdurchläufe in binären Suchbäumen (10 Punkte) – durchführen, analysieren
Aufgabe 3: Einfügen in AVL-Bäumen (12 Punkte) – durchführen
Aufgabe 4: Ein Algorithmus (10 Punkte) – durchführen, analysieren, $O$-Notation
Aufgabe 5: Quicksort (10 Punkte) – durchführen
Aufgabe 6: Dijkstra-Algorithmus (9 Punkte) – durchführen
Aufgabe 7: Ein Baumalgorithmus (26 Punkte) – Eigenschaften, C/C++, $O$-Notation

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!

Zugelassene Hilfsmittel: Keine

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

## Aufgabe 1 — 7 Punkte

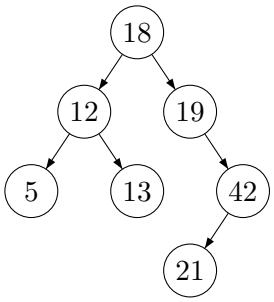
(2+5 = 7 Punkte) **Binäre Suche, Interpolationssuche:** Gegeben ist ein Array mit den Werten (5,6,8,10,23,26,29).

- (2 Punkte) Geben Sie an, welche Elemente bei Verwendung der **binären Suche** beim ersten Vergleich gefunden werden können.
- (5 Punkte) Geben Sie an, welche Elemente bei Verwendung der **Interpolationssuche** beim ersten Vergleich gefunden werden können.

  Geben Sie jeweils eine kurze Begründung an (1–2 Sätze genügen).

## Aufgabe 2 — 10 Punkte

(2+2+2+4 = 10 Punkte) **Binäre Suchbäume:** Gegeben ist folgender binäre Suchbaum:




- (2+2+2 Punkte) Geben Sie die Pre-, In- und Postorderreihenfolge des obigen binären Suchbaums an.
- (4 Punkte) Wie verändert sich die Preorderreihenfolge, wenn Sie ein weiteres Element in den binären Suchbaum einfügen?

## Aufgabe 3 — 12 Punkte

(12 Punkte) Fügen Sie in einen anfangs leeren **AVL-Baum** folgende Zahlen in dieser Reihenfolge ein: 3, 8, 9, 7, 4, 1, 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 — 10 Punkte

(4+4+2 Punkte) **Ein Algorithmus:** Gegeben ist folgendes C++-Programmfragment:

```
const int n=7;
int a[n]={9,7,2,6,3,1,4};
int i,j;
int c=0;
for(i=1;i<n,i++) {
  for(j=0;j<i;j++) {
    if (a[j]>a[i]) {
      c=c+1;
    }
  }
}
```

- (4 Punkte) Welchen konkreten Wert hat die Variable `c` nach Ausführung des obigen Programmfragments?
- (4 Punkte) Welchen Wert hat die Variable in Abhängigkeit der Belegung des Feldes `a[]`? Begründen Sie Ihre Antwort.
- (2 Punkte) Wie groß ist der Aufwand des Programmfragments in Abhängigkeit von `n` in $O$-Notation?

## Aufgabe 5 — 10 Punkte

(10 Punkte) Bei **Quicksort** wird als Pivot-Element das Element in der Mitte gewählt (Index `(l+r)/2`, hier das 11-te der 21 Elemente). Folgendes Feld soll mit Quicksort sortiert werden. Führen Sie die erste Iteration bei Quicksort durch (also das Wandern der Zeiger mit Vertauschungen bis die beiden Zeiger übereinander hinweg gelaufen sind):

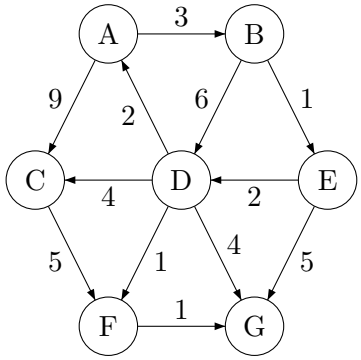
70 22 65 44 16 56 25 9 31 46 49 75 29 88 94 15 19 28 23 77 39

Geben Sie an, welche Vertauschungen durchgeführt werden und welche beiden rekursiven Aufrufe am Ende erfolgen.

## Aufgabe 6 — 9 Punkte

(9 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 E mit d(E)=0 bekannt. Im Dijkstra-Algorithmus wird in jeder Iteration ein Knoten aus der Randmenge R entfernt. 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“ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 0 | $\infty$ | $\infty$ |
| 1. | $\{E\}$ | | | | | | | | |
| 2. | | | | | | | | | |
| 3. | | | | | | | | | |
| 4. | | | | | | | | | |
| 5. | | | | | | | | | |
| 6. | | | | | | | | | |
| 7. | | | | | | | | | |
| 8. | | | | | | | | | |

## Aufgabe 7 — 26 Punkte

(2+1+2+2+16+3 = 26 Punkte) **Ein Baumalgorithmus:** In dieser Aufgabe werden wir ein einfaches Programm entwerfen, das zu einem binären Suchbaum und einem Parameter $k$ das $k$-te Element in diesem Suchbaum bestimmt, und in C/C++-Code umsetzen. (Hinweis: die Fragen weisen einen möglichen Weg, die C/C++-Funktion zu realisieren – sie können die Programmieraufgabe aber auch unabhängig von Ihren vorherigen Antworten lösen.)

Nehmen Sie an, Sie kennen die Anzahl $nl$ der Knoten im linken Unterbaum des Suchbaums.

- (2 Punkte) In welchem mathematischen Kleiner-Größer-Gleich-Verhältnis stehen $nl$ und $k$, wenn das $k$-te Element sich im linken Unterbaum befindet? Das wievielte Element ist es dann, wenn man nur den linken Unterbaum betrachtet?
- (1 Punkte) In welchem mathematischen Kleiner-Größer-Gleich-Verhältnis stehen $nl$ und $k$, wenn das $k$-te Element die Wurzel des Baumes ist?
- (2 Punkte) In welchem mathematischen Kleiner-Größer-Gleich-Verhältnis stehen $nl$ und $k$, wenn das $k$-te Element sich im rechten Unterbaum befindet? Das wievielte Element ist es dann im rechten Unterbaum?
- (2 Punkte) Geben Sie eine C/C++-Datenstruktur für einen binären Suchbaum an. Der Datentyp für eine Variable, die auf die Wurzel eines binären Suchbaums zeigt, soll dabei `bstPtr` heißen.

- (16 Punkte) Schreiben Sie eine C/C++-Funktion `void k_tes_Element(const bstPtr root, const int k)`, die als Parameter den Zeiger auf die Wurzel eines binären Suchbaums und eine Zahl $k$ bekommt. Die geforderte Ausgabe ist:
  - Ist der Baum leer, so soll die Meldung „`Baum ist leer.`“ ausgegeben werden.
  - Ist das eingegebene $k \leq 0$ oder größer als die Anzahl der Knoten im Baum, so soll die Meldung „`Parameter k ungültig!`“ ausgegeben werden.
  - Ansonsten soll die Meldung „`Das k-te Element im Baum ist die x`“ ausgegeben werden, wobei `k` und `x` durch die entsprechenden Werte zu ersetzen sind.

  Zu dem Baum aus Aufgabe 2 soll mit Parameter `k=6` die Ausgabe also „`Das 6-te Element im Baum ist die 21`“ lauten.
- (3 Punkte) Skizzieren Sie den Fall, für den Ihr Algorithmus möglichst lange braucht und geben Sie die Laufzeit in $O$-Notation an.

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

Hinweis: Sie werden zusätzlich zur Funktion vermutlich ein oder zwei separate rekursive Hilfsfunktionen schreiben müssen. Es lässt sich zwar auch alles in einer Funktion integrieren, dies ist aber deutlich schwieriger umzusetzen.

# Lösungshinweise – Wiederholungsklausur in Informatik II – Lewandowski

## Aufgabe 1 — Binäre Suche, Interpolationssuche

Bei Binärer Suche wird stets in der Mitte des Feldes gesucht, es kann beim ersten Vergleich nur das mittlere Element gefunden werden (hier die 10).

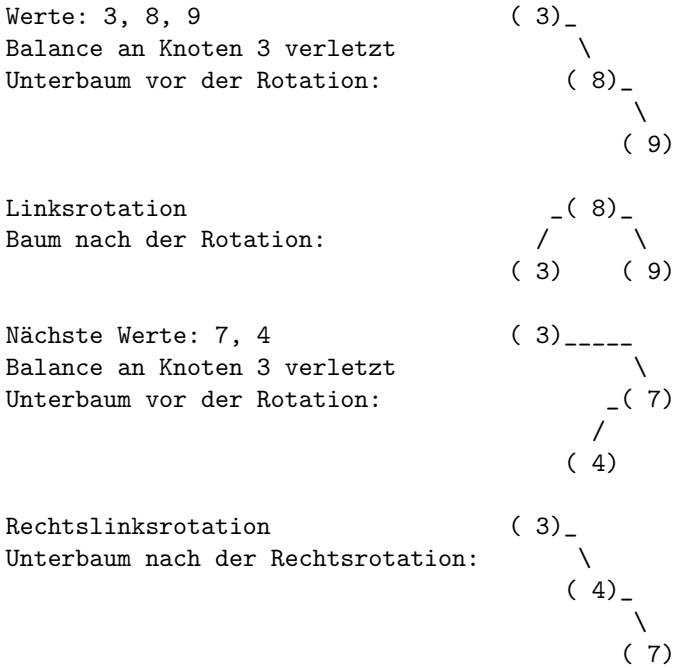
Bei Interpolationssuche schätzen wir ab, wo das gesuchte Element stehen könnte. Prinzipiell kann jedes Element beim ersten Versuch gefunden werden (z.B. im Array (5,9,13,17,21,25,29)). Das erste und letzte Element werden stets beim ersten Versuch gefunden. Es wird jeweils am Index $m = l + (s - a[l]) \cdot (r - l)/(a[r] - a[l])$ gesucht, im Beispiel ist die Suche nach 5, 23, 26 und 29 beim ersten Vergleich erfolgreich.

## Aufgabe 2 — Binäre Suchbäume

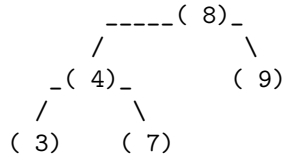
Preorder (erst Wurzel, dann linker, dann rechter Unterbaum): 18 12 5 13 19 42 21
Inorder (erst links, dann Wurzel, dann rechts, stets sortiert): 5 12 13 18 19 21 42
Postorder (erst links, dann rechts, dann Wurzel): 5 13 12 21 42 19 18

Ein neues Element wird stets als Blatt eingefügt. An der Stelle der Preorderreihenfolge, an der zuvor ein leerer Unterbaum war, wird nun das neue Element ausgegeben. Die Reihenfolge aller Elemente bleibt also identisch, das neue Element steht entweder – wenn es ein linkes Kind ist – direkt hinter dem Elter oder hinter dem letzten Knoten der Preorder des linken Unterbaums.

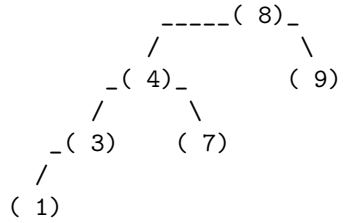
## Aufgabe 3 — AVL-Baum



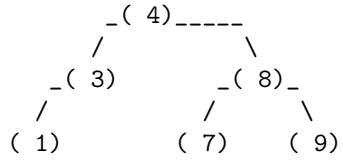
Baum nach der Linksrotation:



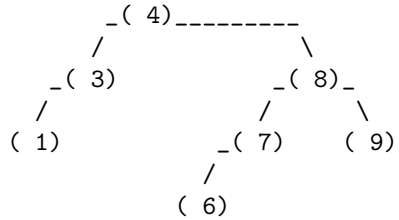

Nächster Wert: 1
Balance an Knoten 8 verletzt
Unterbaum vor der Rotation:




Rechtsrotation
Baum nach der Rotation:




Nächster Wert: 6




## Aufgabe 4 — Ein Algorithmus

Die Variable `c` hat danach den Wert 16. Es wird der Fehlstand (Inversionszahl) des Arrays `a[]` berechnet. Das Programm durchläuft alle Indexpärchen und erhöht den Zähler `c`, wenn `a[i]` und `a[j]` in der falschen Reihenfolge stehen.

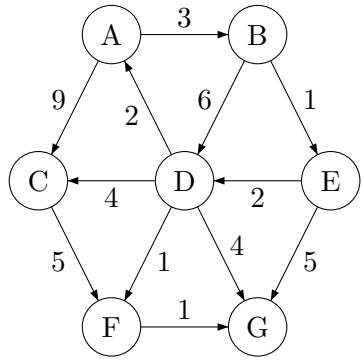
Das Programm besteht aus zwei ineinander geschachtelten Schleifen, die beide bis zu $n$ Durchläufe haben, der Aufwand beträgt $O(n^2)$ – dies hätte als Antwort genügt. Die genauere Abschätzung liefert, dass die `if`-Abfrage $\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = n(n-1)/2$ mal ausgeführt wird – also auch die genauere Abschätzung liefert $O(n^2)$.

## Aufgabe 5 — Quicksort

70 22 65 44 16 56 25 9 31 46 49 75 29 88 94 15 19 28 23 77 39

Als Pivot-Element wird die 49 gewählt. Es werden die 70 mit der 39 getauscht, dann die 65 mit der 23, dann die 56 mit der 28, dann die 49 mit der 19, dann die 75 mit der 15. Der linke Zeiger bleibt dann bei der 88 stehen, der rechte wandert bis zur 28. Die Zeiger sind übereinander hinweggelaufen. Die Parameter der rekursiven Aufrufe sind (linker Rand, rechter Zeiger) (also (0,12)) und (linker Zeiger, rechter Rand) (also (13,20)).

## Aufgabe 6 — Dijkstra




| | Menge R | Erg. von del_min(R) | berechnete Entfernungen A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|---|---|
| 0. | $\emptyset$ | „Init.“ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 0 | $\infty$ | $\infty$ |
| 1. | {E} | E | $\infty$ | $\infty$ | $\infty$ | 2 | 0 | $\infty$ | 5 |
| 2. | {D,G} | D | 4 | $\infty$ | 6 | 2 | 0 | 3 | 5 |
| 3. | {A,C,F,G} | F | 4 | $\infty$ | 6 | 2 | 0 | 3 | 4 |
| 4. | {A,C,G} | A | 4 | 7 | 6 | 2 | 0 | 3 | 4 |
| 5. | {B,C,G} | G | 4 | 7 | 6 | 2 | 0 | 3 | 4 |
| 6. | {B,C} | C | 4 | 7 | 6 | 2 | 0 | 3 | 4 |
| 7. | {B} | B | 4 | 7 | 6 | 2 | 0 | 3 | 4 |
| 8. | $\emptyset$ | „fertig“ | 4 | 7 | 6 | 2 | 0 | 3 | 4 |

In den Zeilen 4 und 5 haben A und G dieselbe Entfernung 4, diese beiden Knoten hätten auch in umgekehrter Reihenfolge bearbeitet werden können.

## Aufgabe 7 — Ein Baumalgorithmus

Hat der linke Unterbaum $nl$ Elemente, dann ist $k \leq nl$, wenn das $k$-te Element sich im linken Unterbaum befinden soll. Ist $k = nl + 1$, so ist es die Wurzel; ist $k > nl + 1$, so befindet sich das Element im rechten Unterbaum.

Es bietet sich an, hier zunächst einen einfachen Durchlauf zur Bestimmung der Anzahl der Knoten des Baumes zu programmieren. Die eigentliche Suchfunktion lässt sich dann leicht rekursiv über obige Eigenschaften finden.

```
struct bst {
  int value;
  bst *left, *right;
};
typedef bst* bstPtr;

int anz(const bstPtr root) { // Durchlauf zur Bestimmung der Anzahl der Knoten
  if (root==NULL) {
    return 0;
  } else {
    return anz(root->left)+1+anz(root->right);
  }
}

int get_k_tes_Element(const bstPtr root, const int k) { // rekursive Hilfsfunktion
  if (root==NULL) {
    return 0; // bei sachgemaesser Nutzung kommt man hier nie hin
  } else {
    int al=anz(root->left);
    if (k<=al) {
      return get_k_tes_Element(root->left,k);
    } else if (k==al+1) {
      return root->value;
```

```
    } else {
      return get_k_tes_Element(root->right,k-al-1);
    }
  }
}

void k_tes_Element(const bstPtr root, const int k) {
  if (root==NULL) {
    cout << "Baum ist leer." << endl;
  } else if (k<1) {
    cout << "Parameter k ungültig!" << endl; // k zu klein
  } else if (anz(root)<k) {
    cout << "Parameter k ungültig!" << endl; // k zu groß
  } else {
    cout << "Das " << k << "-te Element ist die " << get_k_tes_Element(root,k) << endl;
  }
}
```

Der Aufwand für die Bestimmung der Anzahl der Knoten ist $O(n)$. Diese Hilfsprozedur wird dann besonders oft aufgerufen, wenn der binäre Suchbaum zu einer (linksseitigen) Liste entartet ist – also ein Baum, in den die Elemente in absteigender Reihenfolge eingefügt wurden. Wird hier das erste Element gesucht, so sind zunächst $n-1$ Elemente im linken Unterbaum. In diesem wird weitergesucht; in dessen linkem Unterbaum sind nun $n-2$ Elemente usw. Der Aufwand ist auch hier die Summe der natürlichen Zahlen, also quadratisch ($O(n^2)$).

Man kann das Programm auch so schreiben, dass man einen Inorderdurchlauf programmiert und dabei jedes Element mitzählt (vergleiche Lösungshinweise zur Suche nach dem zweitgrößten Element in der Klausur vom 24.7.09) – der Aufwand ist dann auch im schlechtesten Fall linear ($O(n)$).