Lehrstuhl für Informatik III
Prof. Dr.-Ing. M. Nagl

Klausur
Grundgebiete der Informatik I,II

(DPO98)

Datum: 24. 02. 2004

Teil II: Algorithmen und Programmiertechniken

Name: *(in Druckschrift)*	
Matr. Nr.:	
Unterschrift	

Aufgabe	Max. Punkte	Korrektur		Einsichtnahme	
		Punkte	Kürzel	Punkte	Kürzel
II.1	10				
II.2	10				
II.3	20				
II.4	20				
Σ	60				

Aufgabe II.1 Wissensfragen *(10 Punkte)*

Kreuzen Sie bei den folgenden Aufgaben die jeweils richtige Aussage an, bzw. beantworten Sie die Fragen in einem Satz.

Das Ankreuzen falscher Aussagen führt zu Abzügen entsprechend der Punktezahl der betreffenden Frage. Wenn Sie keine der Aussagen ankreuzen, wird die entsprechende Frage mit 0 Punkten gewertet. Insgesamt können 0 Punkte für die Aufgabe B.1 nicht unterschritten werden.

- “Call by Reference” bezeichnet einen Parameterübergabe-Mechanismus, bei dem ... *(1 Punkt)*
  - ☐ der Wert des Aktualparameters in den Formalparameter kopiert wird.
  - ☐ der Wert des Formalparameters den des Aktualparameter verändert.
- Was kann mit Referenz-Parametern passieren, wenn bei der Ausführung einer Prozedur ein Laufzeitfehler auftritt und dieser abgefangen wurde? *(1 Punkt)*

- Wie ist die Lebensdauer einer Variablen definiert? *(1 Punkt)*
  - ☐ nur innerhalb des Blocks, in der sie deklariert ist, und in den umgebenden Blöcken, ggfs. nicht sichtbar aufgrund weiterer Deklarationen mit gleichem Bezeichner
  - ☐ nur im entsprechenden Block, weder in den enthaltenen noch in umgebenden Blöcken
  - ☐ nur im entsprechenden Block und den darin enthaltenen Blöcken, ggfs. nicht sichtbar aufgrund weiterer Deklarationen mit gleichem Bezeichner.
- Definieren Sie kurz die Sichtbarkeit von Variablen. *(1 Punkt)*

- Welche Besonderheit kennzeichnet ein stabiles Sortierverfahren? *(1 Punkt)*
  - ☐ Das Feld ist anschließend absteigend sortiert.
  - ☐ Die Reihenfolge von Elementen mit gleichem Schlüssel wird nicht verändert.
  - ☐ Stabile Sortierverfahren sind für die Sortierung auf stabilen Speichermedien (Bandlaufwerken) besonders gut geeignet.
- Erläutern Sie kurz, bei welcher Art von Daten stabile Sortierverfahren nützlich sind. *(1 Punkt)*

- Das Sortierverfahren “Sortieren durch Einfügen” hat den Aufwand $O(n^2)$. Für welchen Fall gilt dies? *(2 Punkte)*

  - ☐ nur für den besten Fall
  - ☐ nur für den durchschnittlichen Fall
  - ☐ für alle Fälle
  - ☐ für alle Fälle ausser dem besten

- Wodurch unterscheiden sich abstrakte Datentypmodule (ADTs) von abstrakten Datenobjektmodulen (ADOs). *(1 Punkt)*

  - ☐ ADTs stellen einen abstrakten Typ bereit, den ein Verwender noch implementieren muss. Bei ADOs ist dies nicht mehr nötig.
  - ☐ Ein ADT definiert einen abstrakten Typ, von dem mehrere Instanzen erzeugt werden können. Ein ADO stellt nur genau eine Instanz eines Typs zur Verfügung.

- Charakterisieren Sie kurz das Datenabstraktionsprinzip. *(1 Punkt)*

Aufgabe II.2 C++-Syntax und -Semantik *(10 Punkte)*

(a) Im folgenden Programm sind insgesamt 3 Syntaxfehler versteckt. Finden Sie diese und ordnen Sie sie in die Kategorien kontextfreie und kontextsensitive Fehler ein. Tragen Sie Ihre Lösung in die nachfolgende Tabelle ein. Geben Sie für jeden Fehler die Zeilennummer, eine kurze Beschreibung des Fehlers und seine Kategorie an. *(3 Punkte)*

```
1   void StraightSelection(int feld[], int start, int ende) {
2       int elem;
3       for (int i = start; i <= ende - 1; i++) {
4         int k = i;
5         elem = feld[i];
6         for (int j = i + 1; j <= ende; j++) {
7             // ...
8         }
9         feld[k] = feld[j];
10        feld[i] = elem;
11      }
12      return 0;
13  }
14  int main()  {
15    int feld[] ={ 'g', 'a', 'i', 'h', 'n', 'l', 'm', 'x' };
16    StraightSelection(feld, 0, 7);
17    return 0;
18
```

Zeile	Beschreibung	ks/kf

(b) Erstellen Sie, basierend auf der folgenden Beschreibung, ein Codefragment. Sie sollen also kein vollständiges Programm erstellen! *(3 Punkte)*

Zu deklarieren sind ein int-Feld namens feld, dass maximal 100 Elemente speichern kann, und zusätzlich weitere Hilfsvariablen. Das Ende des Feldes wird durch den Wert -1 markiert. Summieren Sie alle Elemente des Felds auf und speichern Sie das arithmetische Mittel in der Variablen mittel ab.

Neben den Deklarationen ist lediglich der Algorithmus zum Bilden des arithmetischen Mittels zu realisieren, nicht jedoch das Füllen des Feldes und Abspeichern des Endes des Feldes durch ein Element mit dem Wert -1.

```
// Deklaration der Variablen
```

```
// Aufsummieren der Werte im Feld. Abbruchbedingung beachten.
```

(c) Spielen Sie den unten angegebenen Algorithmus durch und notieren Sie in der danebenstehenden Tabelle die Werte der Variablen x und y nach jedem Funktionsaufruf. *(4 Punkte)*

```
class A {
public:
   virtual void func(int &x, int &y) {
      x = x + y;
      y = y * 2;
   }
};
class B : public A {
public:
   virtual void func(int &x, int &y) {
      x = x - y;
      y = y / 2;
   }
};
int main() {
   A *a = new A();
   B *b = new B();
   int x = 1, y = 2;
   a->func(x, y);   // (1)
   a = b;
   a->func(x, y);   // (2)
   return 0;
}
```

x	y	
1	2	Anfangswerte
		Wert nach Aufruf (1)
		Wert nach Aufruf (2)

Aufgabe II.3 Implementierung des ADT Phonelist *(20 Punkte)*

In dieser Aufgabe soll ein abstraktes Datentypmodul (ADT) Phonelist implementiert werden, mit dem für eine Menge von Mitarbeitern die Verwaltung der von ihnen geführten Telefongespräche organisiert werden kann.

Für jeden Mitarbeiter wird auf der Halde ein Datensatz der Klasse Data verwaltet, der den Namen und die Gesamtlänge der von diesem Mitarbeiter geführten Telefongespräche speichert.

Die Schnittstelle für die vorgegebene Klasse Data ist wie folgt:

```
class Data {
  private:
    char* name;     // Name des Mitarbeiters
    int minutes;    // Gesamtdauer aller Gespräche des Mitarbeiters
  public:
    Data(char* newName, int newMinutes);     // Konstruktor
    // Zugriffsfunktionen:
    char* getName();
    void setMinutes(int newMinutes);
    int getMinutes();
}
```

Der ADT Phonelist stellt Funktionen für das Einfügen und Löschen von Datensätzen für Mitarbeiter, den Zugriff auf diese Datensätze über einen Cursor, das Ändern der Gesamtdauer der Telefongespräche eines Mitarbeiters, das Sortieren der gesamten Datenstruktur.

Der als Klasse realisierte ADT Phonelist hat folgende Schnittstelle:

```
class Phonelist {
  private:
    // [...]

  public:
    Phonelist();
    // Konstruktor

    void add (int newPersonalNo, char* newName, int newMinutes);
    // Erstellt neuen Datensatz mit Datensätzen auf der Halde

    void remove (int personalNo);
    // Löscht Datensatz für angebenene Personalnummer
```

```
    void changeMinutes(int personalNo, int newMinutes);
    // Ändert im Datensatz für die Personalnummer die Gesprächsdauer

    void sort();
    // Sortierfunktion nach Personalnummer

    // Navigation über alle belegten Plätze:
    void first();      // Cursor auf ersten belegten Platz setzen
    void next();       // Cursor auf nächsten belegten Platz setzen
    bool hasNext();    // Test, ob Cursor weitergesetzt werden kann

    int getPersonalNoAtCursor(); // Personalnummer bei Cursor
    Data* getDataAtCursor();     // Datensatz bei Cursor
}
```

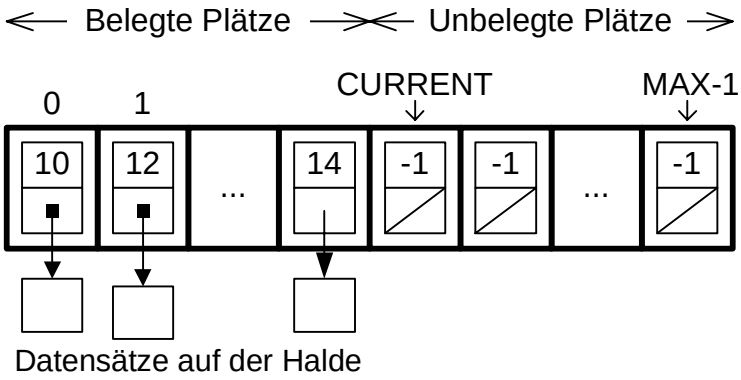
Der ADT Phonelist soll intern mit Hilfe eines Arrays keyEntries mit fest vorgegebener Größe MAX=100 realisiert werden (siehe Bild). Jeder Platz im Array keyEntries ist ein Verbund mit Namen Entry, der die Personalnummer personalNo vom Typ int und einen Zeiger data auf den Mitarbeiter-Datensatz (Klasse Data) mit den weiteren Angaben Name und Dauer der Telefongespräche speichert. Die Datensätze der Mitarbeiter werden auf der Halde abgelegt.

Unbenutzte Array-Plätze sind durch die Verbundwerte {personalNo= -1, data=NULL} gekennzeichnet.

Im Array sollen stets nur die Plätze mit Index i < CURRENT belegt sein. Eine Variable CURRENT vom Typ int soll den Index des ersten unbelegten Platzes speichern.

Die Funktion sort sortiert das Array keyEntries aufsteigend nach der Personalnummer.

Mit first und next kann eine Markierung (Cursor) auf einen beliebigen belegten Platz im Array positioniert werden. Für diesen Platz können Personalnummer und der zugehörige Datensatz mit getPersonalNoAtCursor und getDataAtCursor abgefragt werden.



(a) Ergänzen Sie die nötigen Deklarationen der beschriebenen internen Datenstrukturen im privaten Teil der Schnittstelle von Phonelist. *(3 Punkte)*

```
private:
```

(b) Implementieren Sie den Konstruktor Phonelist. Darin werden die in Teilaufgabe (a) deklarierten Datenstrukturen geeignet initialisiert. *(3 Punkte)*

```
Phonelist::Phonelist(){

}
```

(c) Implementieren Sie die Funktion add für das Einfügen eines Eintrags für einen Mitarbeiter in die Liste. Es darf zur Vereinfachung davon ausgegangen werden, daß Eintragungen mit derselben Personalnummer nicht vorkommen. Einfügen eines Eintrags soll nur möglich sein, solange noch ein unbelegter Platz vorhanden ist. Es wird keine Sortierung beim Einfügen verlangt. *(4 Punkte)*

```
void Phonelist::add(int newPersNo, char* newName, int newMinutes){

}
```

(d) Implementieren Sie die Funktion sort mit Hilfe von Bubblesort. Ergänzen Sie den fehlenden Quellcode, u.a. den Vergleich und die Vertauschung zweier Array-Einträge. Der Tausch findet statt, falls die Personalnummer von e2 kleiner ist als die Personalnummer vom Eintrag e1. *(3 Punkte)*

```
void Phonelist::sort(){
 bool getauscht;
  do{
    getauscht= false;
    // Schleife bis CURRENT vermeidet Sortieren unbelegter Plätze
    for (int i=0; i<CURRENT-1; i++){
      Entry e1= keyEntries[i];
      Entry e2= keyEntries[i+1];
      // [Hier fehlenden Quellcode]









    }
  } while (getauscht);
}
```

(e) Implementieren Sie die Funktion calculateTotalMinutes, die außerhalb der Klasse Phonelist realisiert ist. Diese Funktion berechnet für eine gegegene Telefonliste vom Typ Phonelist die Gesamtsumme der von allen Mitarbeitern geführten Telefongespräche aus den Einträgen in der Telefonliste. Berücksichtigen Sie, daß die Funktion auf die interne Datenstruktur von Phonelist nicht zugreifen kann. *(5 Punkte)*

```
int calculateTotalMinutes(Phonelist phonelist){
  int sum = 0;
  // [Hier Quellcode für Summenberechnung einfügen]









  return sum;
}
```

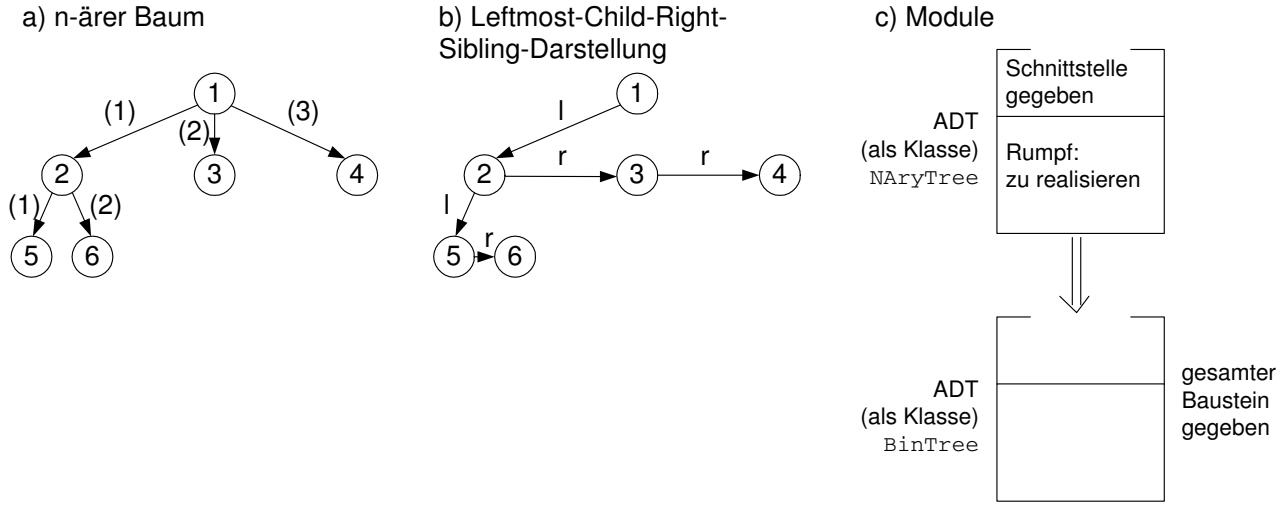
(f) Welches Problem gibt es, wenn für die Realisierung des ADT Phonelist ein Array wie oben benutzt wird? *(1 Punkt)*

(g) Welche alternative Realisierung des ADO Phonelist anstelle von Arrays kennen Sie aus der Vorlesung, mit der direkt beim Einfügen und Löschen von Elementen in die Datenstruktur eine Sortierung vorgenommen werden kann? *(1 Punkt)*

Aufgabe II.4 N-äre Bäume mit Binärbäumen realisieren *(20 Punkte)*

In dieser Aufgabe soll ein ADT für n-äre Bäume `NAryTree` mit Hilfe eines ADT für binäre Bäume `BinTree` realisert werden, wobei letzterer vorgegeben ist. Jeder Baumknoten speichert einen Integerwert. Es ist die aus der Vorlesung bekannte Leftmost-Child-Right-Sibling-Darstellung des n-ären Baums im Rumpf von `NAryTree` anzuwenden. Dabei werden die Knoten eines n-ären Baums als Knoten eines Binärbaums abgespeichert. Das erste Kind eines Knoten ist als linker Nachfolger im Binärbaum (leftmost child) abzuspeichern, die anderen Knoten werden in einer verketteten Liste als rechte Nachfolger im Binärbaum abgelegt (right sibling).

Die folgende Abbildung verdeutlicht Problem und Aufgabenstellung: a) zeigt ein Beispiel eines n-ären Baums, b) seine Leftmost-Child-Right-Sibling-Darstellung, die einem Binärbaum entspricht, c) zeigt die Aufgabenstellung. Dabei soll der ADT `NAryTree` im Rumpf als internen Datenspeicher eine Instanz des ADT `BinTree` verwenden.



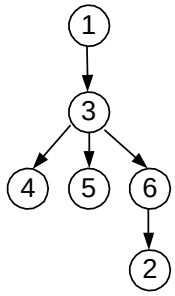
Der folgende Auszug aus der Headerdatei zeigt die Schnittstelle des ADT `NAryTree` und die Deklaration der Instanz des Binärbaums. Ein aktueller Knoten charakterisiert eine entsprechende Stelle im Baum, dieser wird nur von den Navigations-Operationen (`moveTo...`) verändert. Andere Operationen beziehen sich auf die aktuelle Stelle.

Die Schnittstelle des zu verwendenden ADT `BinTree` beschreibt den zu verwendenden Baustein. Auch hier wird der aktuelle Knoten nur von den Navigationsoperationen verändert.

```
class NAryTree {
private:
   BinTree T; // Binärbaum-Instanz
public:
   void createNTRoot(int id);
   // Erzeugt Wurzel mit Wert id; diese ist danach aktueller Knoten
   void addChild(int id);
   // Fügt dem akt. Knoten ein Kind mit dem Wert id rechts von den evtl. existierenden
   // Kindern hinzu. Danach ist der aktuelle Knoten wie vorher gesetzt.
   void moveToNTRoot();
   // Setzt den aktuellen Knoten auf die Wurzel des Baumes.
   void moveToNTParent();
   // Setzt den aktuellen Knoten auf den Vater des aktuellen Knotens.
   void setNTId(int id); //Setzt den Wert des aktuellen Knotens
   int getNTId(); //Liefert den Wert des aktuellen Knotens
   ...
};
```

```
class BinTree {
public:
   void createRoot(int id);
   // Erzeugt ein Wurzelelement mit Wert id und setzt den Cursor auf die Wurzel.
   void setLeftChild(int id);
   void setRightChild(int id);
   // Fügt dem aktuellen Knoten ein linkes bzw. rechtes Kind mit dem Wert id hinzu.
   void setId(int id); //Setzt den Wert des aktuellen Knotens
   int getId(); //Liefert den aktuellen Knoten
   void moveToRoot();
   // Setzt den aktuellen Knoten auf die Wurzel.
   void moveToParent();
   // Setzt den aktuellen Knoten auf den Vater des aktuellen Knotens.
   // Darf nicht für die Wurzel aufgerufen werden.
   bool isRoot();
   // Liefert true, wenn der aktuelle Knoten die Wurzel des Baumes ist.
   bool isRightChild();
   // Liefert true, wenn der aktuelle Knoten das rechte Kind seines Vaters ist.
   // Darf nicht für die Wurzel aufgerufen werden.
   bool moveToLeftChild();
   bool moveToRightChild();
   // Setzt den aktuellen Knoten auf das linke bzw. rechte Kind des aktuellen Knotens.
   // Darf nur angewendet werden, wenn das entsprechende Kind existiert.
   bool hasLeft();
   bool hasRight();
   // Liefert true, wenn der aktuelle Knoten ein linkes bzw. rechtes Kind hat.
   ...
};
```

(a) Gegeben ist der folgende n-äre Baum. Geben Sie die Leftmost-Child-Right-Sibling-Darstellung an. *(3 Punkte)*



(b) Implementieren Sie die Operation `createNTRoot` von `NAryTree`, die die Wurzel eines Baumes mit Wert `id` erzeugt. Sie können davon ausgehen, dass noch keine Wurzel vorhanden ist. *(2 Punkte)*

```
void NAryTree::createNTRoot(int id) {




}
```

(c) Implementieren Sie die Operation `moveToNTParent` von `NAryTree`, die zum Vater navigiert. Dieser ist dann der aktuelle Knoten. Verwenden Sie hierzu die Navigationsoperationen des Binärbaums. Hat der aktuelle Knoten keinen Vater, soll der aktuelle Knoten nicht verändert werden. *(4 Punkte)*

```
void NAryTree::moveToNTParent() {




}
```

(d) Implementieren Sie die Operation addChild von NAryTree, die dem aktuellen Knoten ein Kind mit dem Wert id rechts von den evtl. existierenden Kindern hinzufügt. Beachten Sie, dass der aktuelle Knoten nach Ausführung der Operation wie vorher gesetzt ist. Sie dürfen die Operation moveToNTParent von NAryTree verwenden. *(7 Punkte)*

```
void NAryTree::addChild(int id) {




}
```

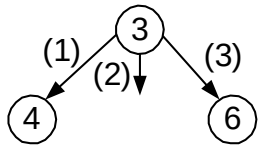
(e) Nach der Leftmost-Child-Right-Sibling-Realisierung für n-äre Bäume, wie in dieser Aufgabe angewendet, sind die Kinder eines Knotens direkt verkettet. Soll beispielsweise ein 3. Kind eingefügt werden, müssen die beiden Vorgängerkinder bereits alle vorhanden sein.

Die Schnittstelle des Bausteins sei nun um folgende Funktionen erweitert:

```
void setChildAt(int id, unsigned int pos);
// Trägt am aktuellen Knoten an der Stelle pos ein Kind mit dem
// Wert id ein. Die Kinder vor pos müssen nicht existieren.

bool moveToChild(unsigned int pos);
// Setzt den aktuellen Knoten auf das Kind des aktuellen Knotens
// an der Stelle pos. Gibt es dieses Kind nicht, ist der Rückgabewert
// false und der aktuelle Knoten verändert sich nicht, sonst true.
```

Die folgende Abbildung verdeutlicht diese Sicht, der Knoten mit dem Wert `3` hat zwei Nachfolger, mit den Werten `4` als erstes und `6` als drittes Kind. Ein zweites Kind gibt es nicht.



Nennen Sie eine Möglichkeit, die Realisierung von `NAryTree` so zu ändern, dass die oben angegebenen Operationen implementiert werden können. Es soll weiterhin das Modul `BinTree` verwendet werden. Es gibt zwei naheliegende Möglichkeiten. Es genügt, wenn Sie eine davon angeben.

Beschreiben Sie kurz Ihren Vorschlag. Skizzieren Sie, wo in der Realisierung Änderungen zu machen sind, bzw. wie die neuen Operationen zu realisieren sind. Sie brauchen keinen Programmcode anzugeben. Die Änderungen beziehen sich evtl. auf beide Bausteine `NAryTree` und `BinTree`. *(4 Punkte)*