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

Klausur
Grundgebiete der Informatik 2

(DPO98/04)

Datum: 19. 03. 2007

Algorithmen und Programmiertechniken

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

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

Aufgabe 1 Syntax und Semantik von C++ *(10 Punkte)*

(a) Finden Sie in der folgenden Prozedur BubbleSort die drei Syntaxfehler und bestimmen Sie, ob diese kontextfreie oder kontextsensitive Fehler sind. *(3 Punkte)*

```
1     void BubbleSort(int zahlen[], int anzahl){
2         bool getauscht;
3         do{
4             int hilf;
5             getauscht=false;
6             for(int i=0; i < anzahl-1; i++){
7                 if(zahlen[i] > zahlen[i+1]){
8                     hilf = zahlen[i];
9                     zahlen[i] = zahlen[i+1];
10                     zahlen[i+1] = hilf;
11                    getauscht = true,
12                }
13            }
14        } while(getauscht);
15        hilf = 0;
16        return 0;
17    }
```

Zeile	Beschreibung	ks/kf

(b) Spielen Sie das nachfolgende C++-Programm durch. Geben Sie die Werte für die Variablen x, y und z der main-Funktion an, nachdem die Prozedur f zum ersten und zum zweiten mal aufgerufen wurde. *(3 Punkte)*

```
void f(int a, int *b, int &c){
    a = (*b) + c;
    (*b) = a + c;
    c = a + (*b);
}

int main (){
    int x = 1, y = 2, z = 3;
    //1. Aufruf der Methode f
    f(x, &y, z);
    //2. Aufruf der Methode f
    f(x, &y, z);
    return 0;
}
```

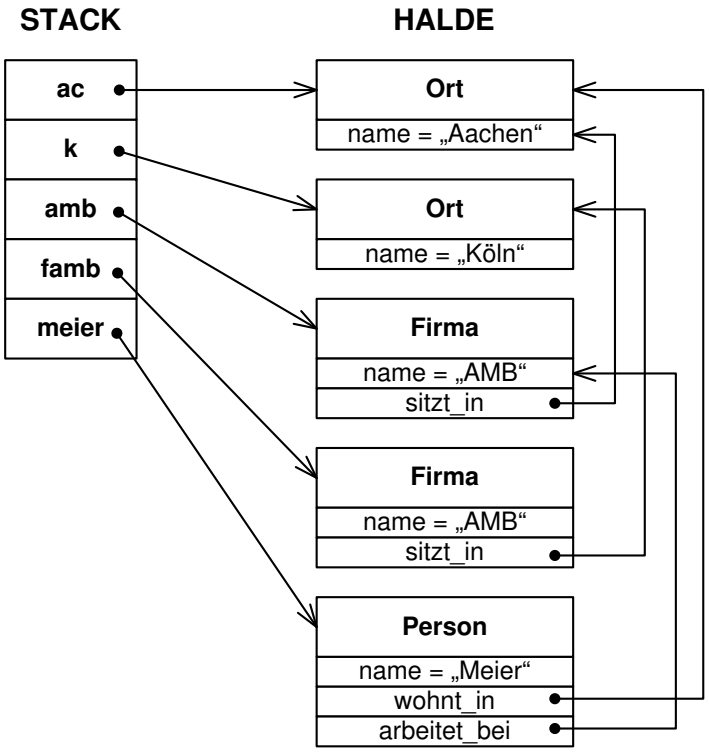
	x	y	z
Anfangswerte	1	2	3
Werte nach 1. Aufruf			
Werte nach 2. Aufruf			

(c) Beschreiben Sie den Unterschied zwischen den Parameterübergabemechanismen Call-by-Value und Call-by-Reference. Geben Sie für jeden Parameter der Prozedur `f` aus der Teilaufgabe (b) den Parameterübergabemechanismus an. *(4 Punkte)*

Aufgabe 2 Zeiger und Objekte *(10 Punkte)*

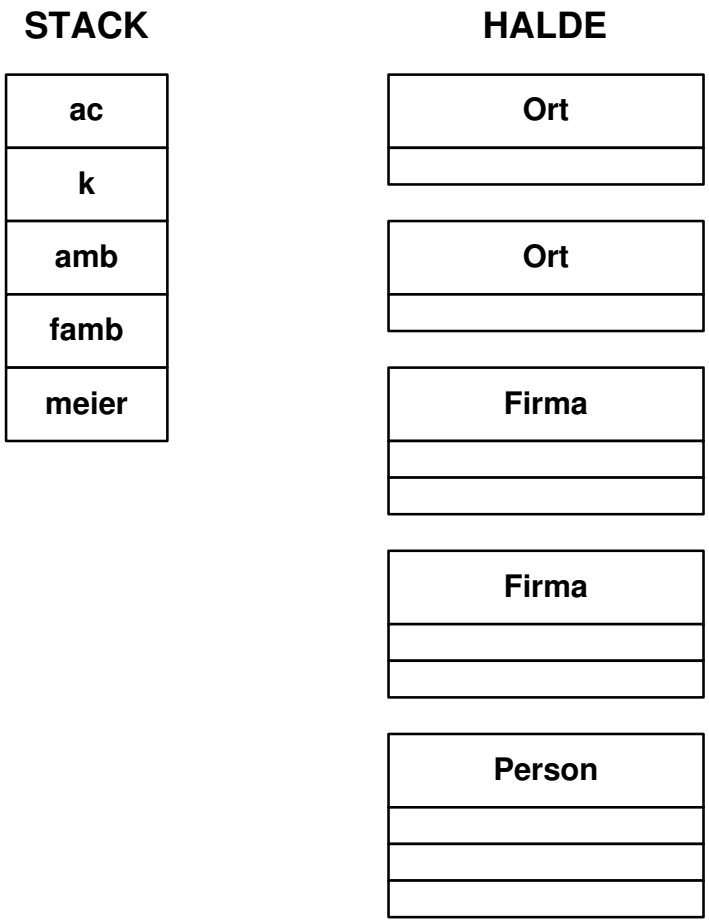
In dieser Aufgabe werden Objekte auf der Halde angelegt, die Instanzen der Verbundtypen Ort, Firma und Person sind. Zusätzlich werden Zeiger auf dem Stack erzeugt, die auf diese Objekte verweisen. Die folgende main-Funktion legt verschiedenen Zeiger und Objekte an, deren Struktur in der nachfolgenden Abbildung (siehe nächste Seite) graphisch dargestellt ist.

```
struct Ort{
    string name;
};

struct Firma{
    string name;
    Ort* sitzt_in;
};

struct Person{
    string name;
    Ort* wohnt_in;
    Firma* arbeitet_bei;
};

int main (){

    Ort* ac = new Ort;
    ac->name = "Aachen";
    Ort* k = new Ort;
    k->name = "Köln";

    Firma* amb = new Firma;
    amb->name = "AMB";
    amb->sitzt_in = ac;
    Firma* famb = new Firma;
    famb->name = "AMB";
    famb->sitzt_in = k;

    Person* meier = new Person;
    meier->name = "Meier";
    meier->wohnt_in = ac;
    meier->arbeitet_bei = amb;

    return 0;
}
```



(a) Zeichnen Sie die Objekt- und Zeigerstruktur nach Ausführung der main-Funktion, wenn die folgenden Zeilen direkt vor dem return-Statement der main-Funktion zusätzlich eingefügt werden. Hierbei sollen die beiden Variablen stra und strk nicht eingezeichnet werden. Ergänzen Sie die folgende Abbildung gemäß der obigen Abbildung und zeichnen Sie die Zeiger und neu erzeugten Objekte ein. *(6 Punkte)*

```
string strk = "Köln";
string stra = "Bad Aachen";

Person* schmitt = new Person;
schmitt->name = "Schmitt";
schmitt->wohnt_in = new Ort;
schmitt->wohnt_in->name = strk;
schmitt->arbeitet_bei = amb;

Firma* famb2 = famb;
meier->arbeitet_bei = famb2;
(*famb2).name = "Filiale AMB";

amb->sitzt_in->name = stra;
schmitt->wohnt_in = schmitt->arbeitet_bei->sitzt_in;
```



(b) Vervollständigen Sie die Prozedur FirmaAusgeben, die den Namen und den Standort einer Firma f auf der Kommandozeile ausgibt. Schreiben Sie die Ausgabe auf, wenn der Prozedur FirmaAusgeben als Aktualparameter amb aus der letzten Teilaufgabe übergeben wird. *(4 Punkte)*

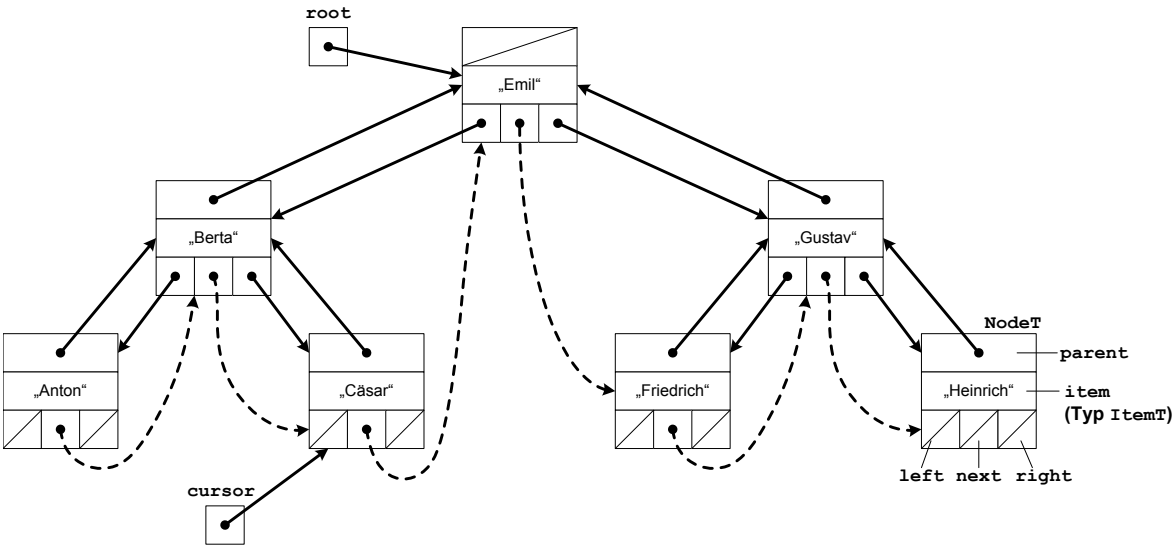
```
void FirmaAusgeben(Firma* f){
    cout << "Firma mit Namen " <<
    cout << " sitzt in " <<
    cout << endl;
}
```

Aufgabe 3 ADT Generische Liste *(20 Punkte)*

In dieser Aufgabe soll ein generisches abstraktes Datentypmodul einer Liste (GenList) implementiert werden. Der öffentliche Teil der Schnittstelle besitzt neben Funktionen zum Suchen und Einfügen/Löschen von Einträgen auch Funktionen zum Iterieren über die Liste:

```
template<class ItemT>
class GenList {
public:
    // Konstruktor für GenList
    GenList();

    // Gibt 'true' zurück, wenn 'item' in der Liste existiert,
    // sonst false.
    bool Contains(ItemT item);

    // Fügt 'item' in der Liste ein und gibt 'true' zurück, wenn 'item'
    // zuvor noch nicht in der Liste existiert hat, sonst 'false'.
    bool Insert(ItemT item);

    // Entfernt 'item' aus der Liste und gibt 'true' zurück, wenn 'item'
    // zuvor in der Liste existiert hat, sonst 'false'.
    bool Remove(ItemT item);

    // Weist 'firstItem' den ersten Eintrag der Liste zu und gibt 'true'
    // zurück, falls Liste nicht leer ist. Andernfalls ist der Rückgabe-
    // wert 'false' und 'firstItem' bleibt unverändert.
    bool ToFirst(ItemT& firstItem);

    // Weist 'nextItem' den nächsten Eintrag der Liste zu und gibt 'true'
    // zurück, falls das Ende der Liste noch nicht erreicht wurde. Andern-
    // falls ist der Rückgabewert 'false' und 'nextItem' unverändert.
    bool ToNext(ItemT& nextItem);

private:
    // diverse Definitionen, siehe Aufgabenteil (a)
    // ...


    // Entfernt den Knoten, auf den 'leaf' verweist, aus dem Suchbaum.
    // Die Prozedur setzt voraus, dass der Knoten existiert und keine
    // Kinder hat (Blattknoten).
    void RemoveLeaf(NodeT* leaf);

};
```

Intern ist die Liste als binärer Suchbaum realisiert. Die Abbildung zeigt einen solchen Suchbaum mit sieben Einträgen, die in diesem Beispiel vom konkreten Typ `string` sind. Zwei Besonderheiten bei der Realisierung sind, dass erstens jeder Baumknoten (Typ `NodeT`) einen Zeiger auf seinen Vater hat und zweitens, dass jeder Baumknoten einen Zeiger (gestrichelte Pfeile) auf seinen Nachfolger gemäß eines Inorder-Durchlaufs durch den Baum hat. Der Zeiger `root` verweist auf die Wurzel des Suchbaums, der Zeiger `cursor` auf den Baumknoten mit dem zuletzt von `ToFirst` bzw. `ToNext` zurückgelieferten Eintrag.



(a) Vervollständigen Sie den privaten Teil der Schnittstelle von `GenList`. Geben Sie die Definition von `NodeT` sowie `root` und `cursor` an. *(4 Punkte)*

(b) Implementieren Sie die Funktion ToNext. Hinweise: Diese Funktion setzt zunächst das Verweisziel des Zeigers cursor (s. privater Schnittstellenteil) von einem Baumknoten zum nächsten Baumknoten entlang des next-Zeigers (gestrichelter Pfeil). Beispiel: Zeigt cursor vor dem Funktionsaufruf auf “Cäsar”, zeigt er anschließend auf “Emil”. Anschließend wird der Eintrag dieses Baumknotens dem per Referenz übergebenen Formalparameter nextItem zugewiesen und true zurückgegeben. Existiert kein nächster Baumknoten, so bleibt nextItem unverändert und es wird false zurückgegeben. *(4 Punkte)*

```
template <class ItemT>
bool GenList<ItemT>::ToNext(ItemT& nextItem) {




}
```

(c) Implementieren Sie die Funktion **Contains**, die genau dann **true** zurückliefert, wenn der in **item** übergebene Eintrag im Baum vorhanden ist. Hinweise: Durchlaufen Sie den Baum von oben nach unten und machen Sie sich dabei die Suchbaumeigenschaft zunutze, die vorschreibt, dass links von einem Baumknoten nur Baumknoten mit kleineren Einträgen und rechts nur Baumknoten mit größeren Einträgen gespeichert sind. *(6 Punkte)*

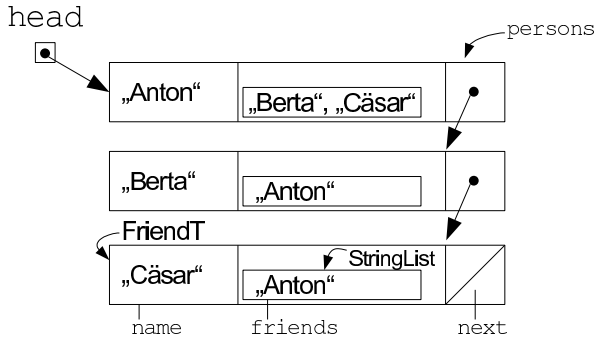
```
template <class ItemT>
bool GenList<ItemT>::Contains(ItemT item) {




}
```

(d) Implementieren Sie die Hilfsfunktion RemoveLeaf, die einen Knoten ohne Kinder (Blattknoten) leaf aus dem Suchbaum entfernt. Hinweise: Hat der Blattknoten, auf den der Parameter leaf verweist, einen Vater, so muss der Zeiger vom Vater auf den Blattknoten mit NULL belegt werden. In jedem Fall muss der Blattknoten von der Halde gelöscht werden. *(6 Punkte)*

```
template <class ItemT>
void GenList<ItemT>::RemoveLeaf(NodeT* leaf) {
    if (leaf->parent == NULL) {
        // Vater ist nicht vorhanden, d.h. leaf war Wurzel
        root = NULL;
    } else {
        // Vater ist vorhanden,
        // d.h. "richtigen" Zeiger des Vaters auf NULL









    // Halde bereinigen




}
```

Aufgabe 4 Soziales Netzwerk *(20 Punkte)*

Ziel dieser Aufgabe ist die Erstellung eines Programms zur Verwaltung sozialer Netzwerke. Das Programm verwaltet die Liste **persons**, die Personen in einer *einfach verketteten, linearen Liste* speichert. Die im privaten Teil der Schnittstelle definierte Variable **head** zeigt auf das erste Element dieser Liste. Einträge sind vom Typ **FriendT** und enthalten den Namen der Person (Typ **string**) sowie die Liste **friends** vom Typ **StringList**. In der Liste **friends** werden Namen von Personen gespeichert, die mit der jeweiligen Person befreundet sind. Für die Realisierung der Liste **friends** soll auf die in Aufgabe 3 entwickelte *generische Liste* zurückgegriffen werden. Freundschaften sollen *symmetrisch* gespeichert werden, d.h. zwei befreundete Personen haben den jeweils anderen Namen in ihrer **friends** Liste eingetragen. Folgende Abbildung zeigt einen möglichen Zustand der Datenstruktur:



Für die Implementierung steht Ihnen in allen Teilaufgaben die Hilfsfunktion **GetPersonByName** (s.u.) zur Verfügung. Diese gibt einen Zeiger auf den entsprechenden Eintrag der **persons** Liste zurück, oder **NULL** falls kein Eintrag mit dem angegebenen Namen exisitiert.

```
class FriendsNetwork {
public:
    FriendsNetwork();
    bool AddPerson(string name);
    bool RemovePerson(string name);
    bool AreFriends(string name1, string name2);
    bool AddFriendship(string name1, string name2);
    int CountFriends(string name);
private:
    /* Private Deklarationen, siehe Aufgabenteil a) */

    FriendT* head;
    FriendT* GetPersonByName(string name);
    void RemoveFromFriends(string name);
};
```

(a) Vervollständigen Sie den privaten Teil der Schnittstelle **FriendsNetwork.h**: Für die Liste **friends** ist der Datentyp **StringList** basierend auf der generischen Liste in Aufgabe 3 geeignet zu definieren. Außerdem ist der Verbundtyp **FriendT** zur Realisierung der verketteten Liste zu vervollständigen. *(3 Punkte)*

```
// Typ für friends Liste




// Typ für persons Liste
struct FriendT {









};
```

(b) Zum Entfernen einer Person aus dem sozialen Netzwerk muss auch ihr Name aus den **friends** Listen der anderen Personen entfernt werden. Implementieren Sie diesen Teil in der privaten Funktion **RemoveFromFriends**. Nutzen Sie dabei die in der Klasse **GenList** definierte **Remove** Funktion. *(4 Punkte)*

```
RemoveFromFriends(string name) {
    // Für alle Personen der persons Liste...




    // ...entferne die durch name angegebene Person aus der friends Liste.




}
```

(c) Das Hinzufügen einer Freundschaft soll in der Funktion AddFriendship erfolgen. Sie können davon ausgehen, dass sich Einträge für beide Personen in der Liste befinden. Eine Freundschaft darf nicht hinzugefügt werden, falls die Personen bereits befreundet sind. Auch soll es nicht möglich sein, eine Person als ihren eigenen Freund einzutragen. Falls eine Freundschaft eingetragen wurde, gibt die Funktion true zurück, andernfalls false. Beachten Sie bei der Implementierung, dass die Freundschaftsbeziehung symmetrisch gespeichert werden soll. *(5 Punkte)*

```
bool FriendsNetwork::AddFriendship(string name1, string name2) {
    // Schon befreundet oder gleiche Person?




    // Zeiger aus der linearen Liste ermitteln




    // Freundschaft eintragen




    // Rückgabewert je nachdem, ob Freundschaft eingetragen wurde


}
```

(d) In der Funktion **CountFriends** sollen Sie ermitteln, wieviele Freunde eine bestimmte Person besitzt. Sie können davon ausgehen, dass sich ein Eintrag für den angegebenen Namen in der Liste befindet. Hinweis: Nutzen Sie die Funktionen **ToFirst** und **ToNext** aus der Klasse **GenList** zum Durchlaufen der **friends** Liste. *(8 Punkte)*

```
int FriendsNetwork::CountFriends(string name) {
    // Zeiger auf Person ermitteln



    // Hat einen Freund?









    // Hat noch mehr Freunde?









    // Wert zurück geben



}
```