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

Klausur
Grundgebiete der Informatik I,II

(DPO98/04)

Datum: 23. 03. 2006

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)*

(a) Was ist der Unterschied zwischen einer `break`- und einer `return`-Anweisung? *(1 Punkt)*

(b) Worin besteht der wesentliche Unterschied zwischen einem Feld und einem Verbund? *(1 Punkt)*

(c) Wodurch unterscheiden sich Funktionen von Prozeduren? *(1 Punkt)*

(d) Was versteht man unter dem Begriff *Aliasing*? *(1 Punkt)*

(e) Skizzieren Sie kurz die grundlegende Idee von *Sortieren durch Auswählen.* *(1 Punkt)*

(f) Welche wesentliche Eigenschaft besitzt ein Baum im Gegensatz zu einem Graphen? *(1 Punkt)*

(g) Worauf muss geachtet werden, wenn man einen Graphen mit Hilfe einer *sequentiellen Kantenliste* vollständig speichern will? *(1 Punkt)*

(h) Wer kann auf den *protected*-Teil der Schnittstelle einer Klasse zugreifen? *(1 Punkt)*

(i) Wofür wird das Schlüsselwort `this` innerhalb von Klassen verwendet? *(1 Punkt)*

(j) Was ist der Unterschied zwischen einem funktionalen Modul und einem Datenobjektmodul? *(1 Punkt)*

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

(a) Gegeben sei das folgende Programm, das den Sortiertalgorithmus “Sortieren durch Auswählen” realisiert.

```
1   #include <iostream>
2   using namespace std;
3
4   void StraightSelection(char feld[], int start, int ende) {
5     char elem;
6
7     for (int i = start; i <= ende - 1; i++) {
8       int k = i;
9       elem = feld[i];
10      for (int j = i + 1; j <= ende; j++) {
11        if (feld[j] < elem) {
12          k = j
13          elem = feld[j];
14        }
15      }
16      feld[k] = feld[j];
17      feld[i] = elem;
18    }
19  }
20
21  int main()  {
22    char feld[] ={ 'b', 'a', 'i', h, 'n', 'l', 'm', 'x' };
23    StraightSelection(feld, 0, 7);
24    for (int i = 0; i < 4, i++) {
25      cout << feld[i];
26    }
27    return 0;
28  }
```

Zeile	Beschreibung	ks/kf

In diesem Programm sind insgesamt 4 Syntaxfehler versteckt. Finden Sie diese und ordnen Sie sie in die Kategorien kontextfreie und kontextsensitive Fehler ein. Geben Sie jeweils neben der Zeilennummer und der Kategorie auch eine kurze Begründung an. Hinweis: In den Zeilen 1 und 2 sind keine Fehler.

(4 Punkte)

(b) In dem folgenden Programm werden die Variablen `x`, `y` und `z` als Aktualparameter an die Prozedur `f` übergeben. Geben Sie den Wert von `x`, `y` und `z` nach der jeweiligen Ausführung der Prozedur `f` an.

(6 Punkte)

```
void f(int a, int *b, int &c) {
   a++;
   *b += 1;
   c = (*b)*a;
}


int main() {
   int x = 1, y = 2, z = 3;

   f(x, &y, z);
   f(x, &y, z);
   f(x, &y, z);

   return 0;
}
```

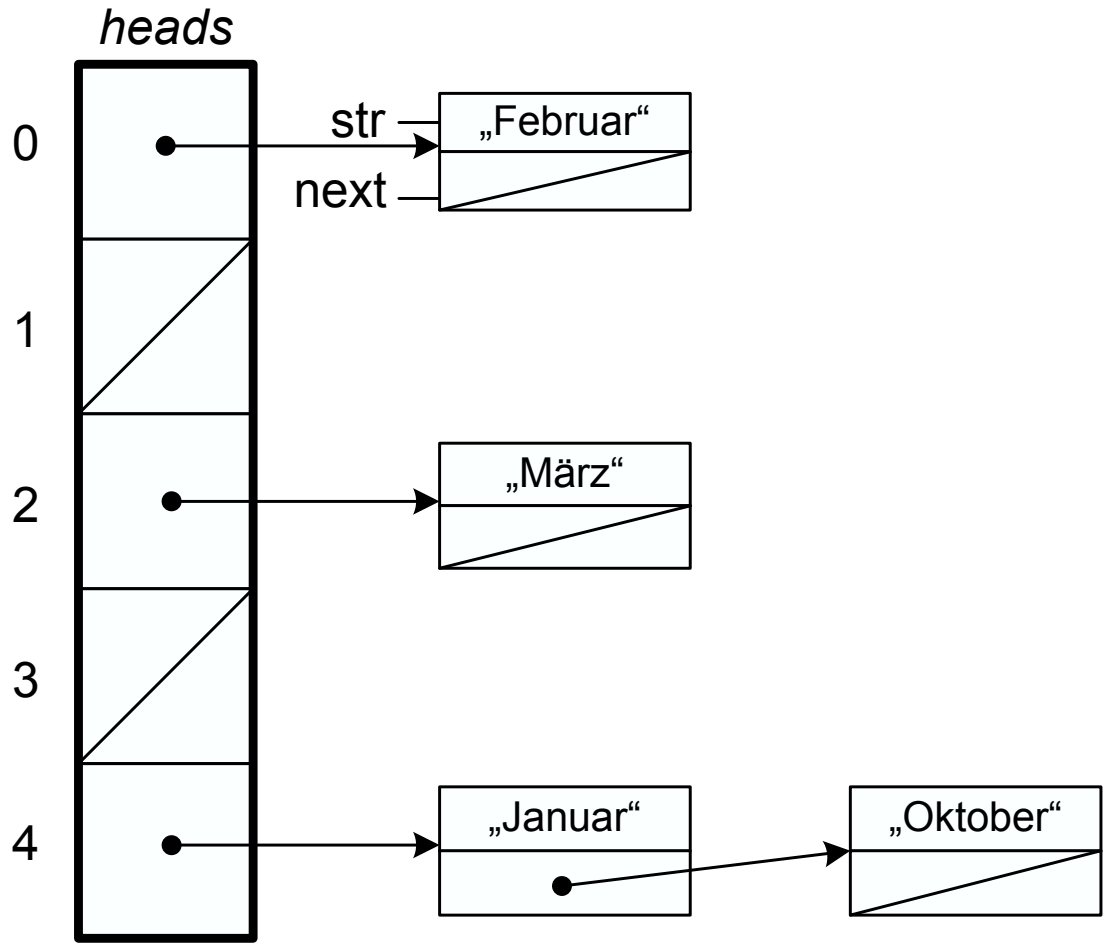
Durchlauf	x	y	z
	1	2	3
1			
2			
3			

Aufgabe II.3 ADT StringSet *(20 Punkte)*

In dieser Aufgabe soll ein ADT für eine Menge realisiert werden, deren Elemente Zeichenketten aus Buchstaben (Strings) sind. Die Strings werden in einer Datenstruktur abgelegt, wie sie unter Teilaufgabe (a) dargestellt ist. Zur Ablage wird Hashing genutzt, d.h. der Speicherort wird über eine Hashfunktion bestimmt. Elemente mit dem gleichen Hashwert werden in einer Liste gespeichert.

In unserem Fall berechnet die Hashfunktion die Position des Anfangsbuchstabens des Strings im Alphabet modulo 5. Die Position von “A” sei dabei 0. Der berechnete Wert `i` wird als Index für ein Feld `heads` verwendet, dessen Elemente Zeiger auf Listenköpfe sind. Wird ein vorhandener String `s` gesucht, so befindet er sich in der Liste, auf die `heads[i]` verweist. Wird ein neuer String `s` der Menge hinzugefügt, so wird er ans *Ende* der Liste hinzugefügt, auf die `heads[i]` verweist.

(a) Ausgehend vom Zustand, wie er in der Abbildung unten zu sehen ist, werden die Strings “Mai” und “Dezember” der Menge hinzugefügt. Vervollständigen Sie die Abbildung. *(2 Punkte)*



Die Menge aus Strings hat folgende Schnittstelle:

```
class StringSet {
public:
    // Erzeugt neue String-Menge
    StringSet();

    // Fügt neuen String zur Menge hinzu
    void Add(string s);

    // Gibt true zurück, gdw. der mit 's'
    // bezeichnete String in der Menge ist
    bool Contains(string s);

    // Setzt den Iterator auf den ersten String
    void MoveToFirst();

    // Gibt true zurück, gdw. ein aktueller String existiert
    bool HasCurrent();

    // Gibt den aktuellen String zurück, sofern existent
    // Vorbedingung: HasCurrent() muss zu true evaluieren
    string GetCurrent();

    // Setzt den Iterator auf den nächsten String. Falls dieser nicht
    // existiert, liefert HasCurrent() nach Aufruf von MoveNext() false.
    void MoveNext();

private:
    // Deklaration von LElem fehlt hier (s. Teilaufgabe b)

    // Anzahl der Listen
    const static int HEADSSIZE = 5;

    // Hashfunktion
    int Hash(string s);

    // Feld mit Zeiger auf Listenköpfe
    LElem* heads[HEADSSIZE];

    // (Iterator) Aktuelle Liste
    int headsPos;

    // (Iterator) Aktuelles Listenelement
    LElem* lePos;
};
```

(b) Geben Sie die im privaten Teil der Schnittstelle von StringSet fehlende Deklaration des Listenelementtyps LElem an. *(2 Punkte)*

```
struct LElem {




};
```

(c) Zum Iterieren über ein StringSet werden zwei private Variablen benötigt. Die ganze Zahl headsPos speichert den Index des Zeigers auf die Liste, in der sich das Listenelement des aktuellen Strings befindet. Der Zeiger lePos verweist auf dieses Listenelement. Implementieren Sie die Prozedur MoveNext(), welches lePos das nächste Listenelement zuweist. Verweist lePos auf das *letzte* Listenelement einer Liste, so wird lePos auf das *erste* Listenelement der nächsten nichtleeren Liste gesetzt. Der Index headsPos muss dabei entsprechend erhöht werden. Sind alle folgenden Listen leer, so ist nach Beendigung der Prozedur lePos == NULL und headsPos == HEADSSIZE. *(7 Punkte)*

```
void StringSet::MoveNext()
{









}
```

(d) Implementieren Sie den Konstrkutor von **StringSet**. Stellen Sie dabei sicher, dass die Elemente von **heads** anfangs alle NULL sind. *(2 Punkte)*

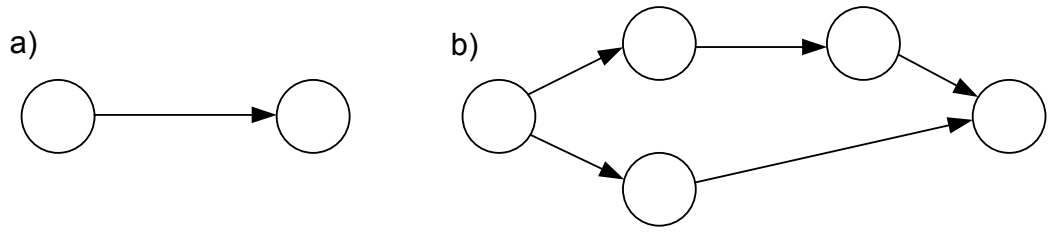
```
StringSet::StringSet()
{




}
```

(e) Implementieren Sie die Funktion **Contains(string s)**, die genau dann **true** zurückgibt, falls der übergebene String **s** in der Menge vorhanden ist. Hinweis: Sie können zwei Strings über den Gleichheitsoperator = auf Gleichheit überprüfen. Die Funktionen/Prozeduren zum Iterieren über die Menge vereinfachen die Implementierung von **Contains** *nicht*. *(7 Punkte)*

```
bool StringSet::Contains(string s)
{









}
```

Aufgabe II.4 ADO für Transportnetze *(20 Punkte)*

In dieser Aufgabe soll, basierend auf einem ADT für Graphen, ein ADO zur Darstellung von Transportnetzen realisiert werden. Ein Transportnetz besitzt genau eine Quelle und eine Senke, die über mehrere gerichtete Kanten und Knoten miteinander verbunden sind. Die folgende Abbildung a) zeigt das initiale Transportnetz, das nur aus Quelle und Senke besteht, die direkt miteinander verbunden sind. Teil b) der Abbildung zeigt ein einfaches Beispiel-Transportnetz.



Das ADO soll die folgende Schnittstelle zum Aufbau von Transportnetzen anbieten:

```
typedef int Knoten;

// Loescht ggf. vorhandene Knoten und Kanten
// und legt dann Quelle und Senke an, die direkt verbunden sind
// (vgl. a) in der Abbildung), und gibt diese in den
// beiden Referenzparametern zurueck.
void TransportnetzInitialisieren(Knoten &quelle, Knoten &senke);

// Fuegt zwischen zwei Knoten einen weiteren ein, wobei eine
// Kante vom ersten zum neuen und eine Kante vom neuen zum zweiten
// Knoten angelegt werden. Die ursprünglich vorhandene Kante zwischen
// den Knoten wird gelöscht. Rückgabewert ist der neue Knoten.
Knoten TransportnetzKnotenEinfuegenZwischen(Knoten erster, Knoten zweiter);

// Teilt einen Knoten in zwei parallele auf, indem ein neuer Knoten
// erzeugt wird und die einlaufenden und auslaufenden Kanten des alten
// Knotens auf den neuen Knoten kopiert werden. Rückgabewert ist
// der neue Knoten.
Knoten TransportnetzKnotenAufteilen(Knoten alt);
```

(a) Zunächst soll das ADO **Transportnetz** angewendet werden: Ergänzen Sie das folgende Hauptprogramm so, dass das Transportnetz aus Abbildung b) erzeugt wird. *(4 Punkte)*

```
int main() {









    return 0;
}
```

Zur Realisierung des ADO **Transportnetz** wird der ADT Graph mit folgender Schnittstelle eingesetzt:

```
class Graph {
   public:
      // Konstruktor, erzeugt einen leeren Graphen
      Graph();

      // loescht alle Knoten und Kanten im Graph
      clear();

      // erzeugt einen neuen Knoten, Rückgabewert ist
      // die dem Knoten automatisch zugeordnetet Nummer
      int addNode();

      // fuegt eine Kante vom Knoten mit der nummer
      // 'from' zum Knoten mit der Nummer 'to' ein
      void addEdge(int from, int to);

      // loescht die durch 'from' und 'to' festgelegte Kante
      void removeEdge(int from, int to);

      // Mit den folgenden Funktionen koennen die Vorgänger bzw.
      // Nachfolger eines Knotens durchlaufen werden.
      // Alle geben -1 zurueck, falls kein Nachfolger/Vorgaenger
```

```
      // mehr vorhanden ist.

      // gibt den ersten Vorgaenger des Knoten mit der Nummer 'node'
      // zurueck und setzt den internen Zaehler für den aktuellen
      // Vorgaenger auf den ersten.
      int firstPredecessor(int node);

      // Gibt den nachsten Vorgaenger des Knoten mit der Nummer 'node'
      // zurueck und erhoeht den internen Zaehler um eins.
      int nextPredecessor(int node);

      // analog fuer die Nachfolger
      int firstSuccessor(int node);
      int nextSuccessor(int node);
private:
   ...
};
```

(b) Geben Sie die private Deklaration der Datenstruktur im Rumpf des ADO **Transportnetz** an. Denken Sie daran, dass sich das ADO auf den ADT **Graph** abstützen soll.
(2 Punkte)

(c) Implementieren Sie die Funktion **TransportnetzInitialisieren**. *(4 Punkte)*

```
void TransportnetzInitialisieren(Knoten &quelle, Knoten &senke) {




}
```

(d) Implementieren Sie die Funktion TransportnetzKnotenEinfuegenZwischen. Sie können davon ausgehen, dass die übergebenen Knoten sowie die Kante zwischen ihnen existieren, brauchen dies also nicht zu überprüfen. *(5 Punkte)*

```
Knoten TransportnetzKnotenEinfuegenZwischen(Knoten erster,
                                            Knoten zweiter) {









}
```

(e) Implementieren Sie die Funktion TransportnetzKnotenAufteilen. Bitte implementieren Sie nur die gekennzeichneten Teile. Sie brauchen nicht zu prüfen, ob der übergebene Knoten existiert. *(5 Punkte)*

```
Knoten TransportnetzKnotenAufteilen(Knoten alt) {
   // Knoten anlegen (bitte implementieren)



   // einlaufende Kanten kopieren (bitte implementieren)
```

```
   // auslaufende Kanten kopieren (bitte _nicht_ implementieren!)
   // ...
   // Rueckgabe (bitte implementieren)



}
```