Matrikelnummer: Name: #:
Hörsaal: Sitzplatznummer:

Prof. Dr.-Ing. K. D. Müller-Glaser

Institut für Technik der Informationsverarbeitung, KIT

Informationstechnik

SS 2012

Probeklausur SS 2012

Mi., 18.07.2012

Hinweise zur Klausur

Hilfsmittel

Zur Prüfung ist nur eine handgeschriebene 2-seitige A4-Formelsammlung zugelassen. Nicht erlaubt sind die Verwendung eines Mobiltelefons und jegliche Kommunikation mit anderen Personen.

Prüfungsdauer

Die Prüfungsdauer beträgt 60 Minuten.

Prüfungsunterlagen

Die Prüfungsunterlagen bestehen aus insgesamt 14 Seiten Aufgabenblättern (diesem Titelblatt, 7 Aufgabenblöcken, 0 zusätzlichen Lösungsblättern)

Bitte kontrollieren Sie vor der Bearbeitung der Aufgaben auf jeder Seite oben Ihren Namen und Ihre Matrikelnummer!

Falls Sie zusätzliche Blätter zur Lösung der Aufgaben benötigen, verwenden Sie die zusätzlichen Lösungsblätter bzw. fragen Sie nach zusätzlichem Papier. Auf jedes zusätzliche Lösungsblatt ist neben dem Namen auch die Aufgabennummer mit einzutragen. Vermeiden Sie das Beschreiben der Rückseiten. Die Verwendung von eigenen Blättern ist nicht erlaubt. Am Ende der Prüfung sind alle 14 Seiten und alle zusätzlichen Lösungsblätter im ausgehändigten Umschlag abzugeben. Verwenden Sie zum Bearbeiten der Aufgaben nur dokumentenechte Schreibgeräte – keinen Bleistift, keine Rotstifte!

Wie verabredet enthält die Klausur mehr Aufgaben als, bei einer richtigen Lösung, zum Erreichen einer sehr guten Note (1,0) notwendig sind (Nutzen Sie die Auswahlmöglichkeit). Bitte beachten Sie: Die mit * gekennzeichneten Teilaufgaben können Sie nicht unabhängig von den anderen Teilaufgaben lösen.

Aufgabe	1	2	3	4	5	6	7		Σ
≈ Gewichtung ca. [%]									100
Ergebnis [P]									

Aufgabe 1 Allgemeine Fragen

Beantworten Sie folgende Fragen:

A) Nennen Sie den Unterschied zwischen Funktionen einer imperativen Programmiersprache und Methoden einer objektorientierten Programmiersprache.

B) Beantworten Sie die folgenden Fragen:

Bitte beachten Sie, dass falsche Antworten einen Punktabzug bedeuten. Negative Punkte werden allerdings nicht auf andere Teilaufgaben übertragen.

Richtig	*Falsch*	
☐	☐	Jeder iterative Algorithmus lässt sich auch als rekursiver Algorithmus darstellen.
☐	☐	InsertionSort ist ein stabiles Sortierverfahren.
☐	☐	InsertionSort hat im Optimalfall eine Komplexität in O(log n)
☐	☐	Im Gegensatz zu einem Array eignet sich eine Liste besonders gut für die binäre Suche.

C) Bringen Sie die folgenden Begriffe in eine logische Reihenfolge und stellen Sie die Beziehungen grafisch dar:

Linker - Headerdateien - Standardbibliotheken - ausführbare Datei - Compiler - Quelldatei - Editor - Objektdatei

D) Beim Simulated-Annealing Algorithmus kann es bei mehrmaligem Ausführen zu unterschiedlichen Endergebnissen kommen. Warum?

Aufgabe 2 C++ Verständnisfragen

A) Gegeben ist der folgende kurze Programmausschnitt. Nehmen Sie an, dass das Programm korrekt und fehlerfrei kompiliert worden ist. Welches sind die Ausgaben auf dem Bildschirm aufgrund der dargestellten Zeilen C++ Programmcode?

```
int main() {
  int i = 5;

  cout << 5 / 2 << endl;
  cout << 5.0 / 2 << endl;
  cout << i++ << endl;
  cout << ++i << endl;
}
```

B) Gegeben ist das folgende C++ Programm. Dieses enthält Programmzeilen, welche sowohl beim kompilieren, als auch zur Laufzeit zu Fehlern führen können. Benennen Sie drei mögliche Fehlerquellen im C++ Programmcode.

```
#include <iostream>

int main() {
  int v = 3;
  array1[5] = {1, 2, 3, 4, 5};
  v = array1[5];
  cout << "Hallo" << Welt;
  return 0;
}
```

1)

2)

3)

C) Welchen Wert speichert die Variable *x* nach Ausführung der folgenden Schleife ?

```
int x = 0;
for( int i = 0; i < 4; x += i++ );
```

x = ☐

D) Finden Sie jeweils einen Fehler in den zwei folgenden Quellcodeabschnitten?

a)
```
class A {
  private:
    string typ;
  public:
    void set( string s ) const {
      typ = s;
    }
    string get() const {
      return typ;
    }
};
```

Antwort:

b)
```
class B {
  private:
    double Arr[5];
  public:
    feld( int i ) {
      return Arr[i];
    }
};
```

Antwort:

G) Richtig oder falsch?

Bitte beachten Sie, dass falsche Antworten einen Punktabzug bedeuten. Negative Punkte werden allerdings nicht auf andere Teilaufgaben übertragen.

i) Ein Zeiger repräsentiert die Adresse und den Typ eines Objekts.

☐ Falsch ☐ Richtig

ii) Der Operator **&&** verknüpft zwei Operanden und gibt **true** zurück, wenn beide Operanden den Wert **true** haben, sonst **false**.

☐ Falsch ☐ Richtig

iii) Der Vorteil einer verketteten Liste gegenüber einem dynamisch angelegten Array ist, dass die verkettete Liste weniger Speicherplatz belegt.

☐ Falsch ☐ Richtig

iv) Der Operator **new** erwartet als Operand den Typ des anzulegenden Objekts.

☐ Falsch ☐ Richtig

Aufgabe 3 Datenstrukturen

A) Nennen Sie 6 unterschiedliche Datenstrukturen.

B) Ein Array wird stets in einem zusammenhängenden Speicherbereich abgelegt.

☐ Richtig ☐ Falsch

C) Schreiben Sie maximal 5 Zeilen C++ Programmcode, um ein **`float`** Array mit insgesamt 12 Werten anzulegen und dieses in einer Schleife mit Werten von 1,00 bis inkl. 1,11 (Abstand: 0,01) zu füllen.

D) Zeichnen Sie ein Nassi-Shneiderman Diagramm, welches einen Algorithmus beschreibt, um die Fakultät einer Zahl zu berechnen. Die Zahl soll dabei von der Tastatur eingelesen werden und vor der Berechnung überprüft werden, ob Sie gültig (größer oder gleich Null) ist. Bei einer negativen Zahl soll eine Fehlermeldung ausgegeben werden, ansonsten soll das Ergebnis in einer Schleife berechnet und ausgegeben werden.

Aufgabe 4 Studentenverwaltung

Das Institut für Technik der Informationsverarbeitung möchte für die Klausur Informationstechnik ein Tool in C++ entwickeln, um die Studenten und ihre Note zu verwalten. Sie haben die Aufgabe, dieses Tool zu programmieren.
Die Klasse, in der die Daten der Studenten gespeichert sind, sieht folgendermaßen aus:

```
class Student {
  public:
    string name;
    short matrikel;
    float note;
    Student* next;
};
```

Da die Anzahl der Teilnehmer an der Klausur nicht vorhergesehen werden kann, sollen die Daten in einer speziellen verketteten Liste gespeichert werden. Die verkettete Liste soll wie in Abbildung 4.1 gezeigt aufgebaut sein. Bitte beachten Sie dabei, dass das letzte Element wieder auf das erste Element der verketteten Liste zeigt und dass kein Zeiger für das Ende der verketteten Liste vorhanden ist.

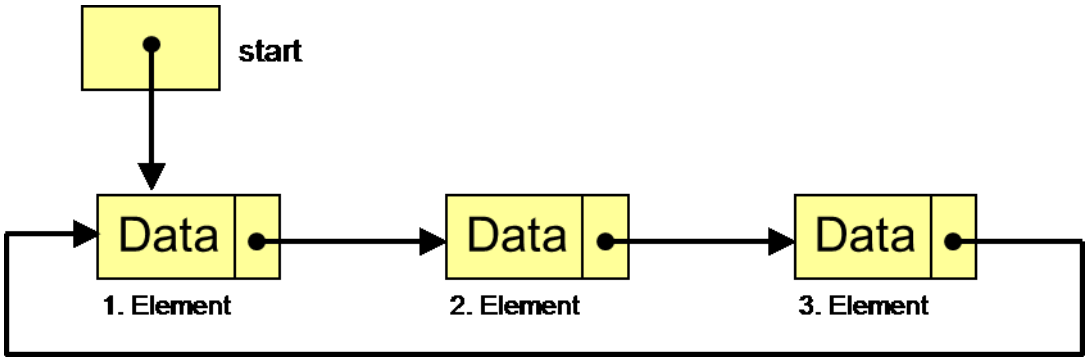


Abbildung 4.1 prinzipieller Aufbau der verketteten Liste

Beim Starten des Programms sollen die Daten aus einer Datei eingelesen werden und die verkettete Liste entsprechend aufgebaut werden. Die folgende Abbildung zeigt einen Ausschnitt aus der Datei. Beantworten Sie die folgenden Teilaufgaben, welche das darstellte Problem in Teilen lösen. Es gilt dabei allgemein, dass eventuell verwendete Variablen vorher angelegt werden müssen.

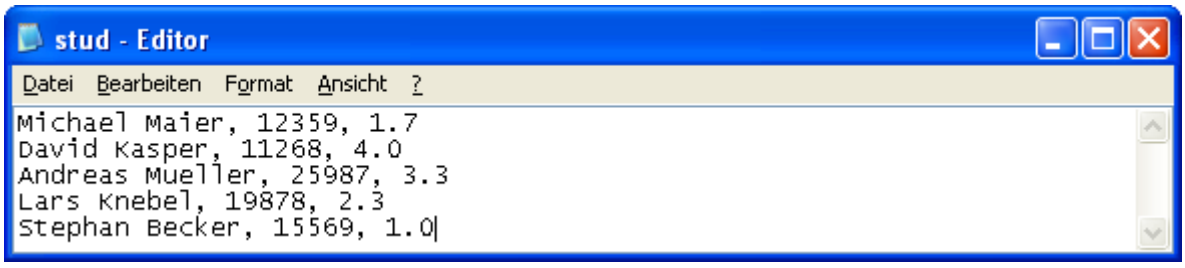


Abbildung 4.2 prinzipieller Aufbau der Datei

A) Geben Sie den Programmcode an, um die oben stehende Datei *stud.txt* zum Lesen zu öffnen. Und geben Sie zwei bis drei weitere Zeilen Programmcode an, mit welchen eine Fehlermeldung ausgegeben wird, wenn das Öffnen der Datei nicht erfolgreich war.

B) Wie würde eine Schleife aussehen, mit welcher die Datei Zeilenweise bis zum Dateiende ausgelesen wird. Maximal 5 Zeilen.

C) Es sei eine Zeile aus der Datei ausgelesen und an die Funktion *zeileAuswerten()* übergeben (*string zeile*). Ergänzen Sie hierzu die folgende Funktion. Diese soll mit Hilfe von Befehlen zur Stringmanipulation den String in der Übergabevariablen *zeile* in seine verschiedenen Bestandteile (Name, Matrikelnummer, Note – alle Varibalentyp String) zerlegen und den entsprechenden gegebenen Variablen zuweisen. Es kann davon ausgegangen werden, dass alle Zeilen das gleiche Format haben und immer alle Werte vorhanden sind, wie in Abbildung 4.2 gezeigt. In den unteren gegebenen Zeilen werden die String-Variablen anschließend, wenn nötig, umgewandelt und dann der Funktion *elementHinzufuegen()* übergeben – siehe Aufgabenteil D).

```
void zeileAuswerten( string zeile ) {
  string name;
  string s_matrikel;
  string s_note;
```

```
  short matrikel = matrikelUmwandeln( s_matrikel );
  float note = noteUmwandeln( s_note );

  elementHinzufuegen( name, matrikel, note );
}
```

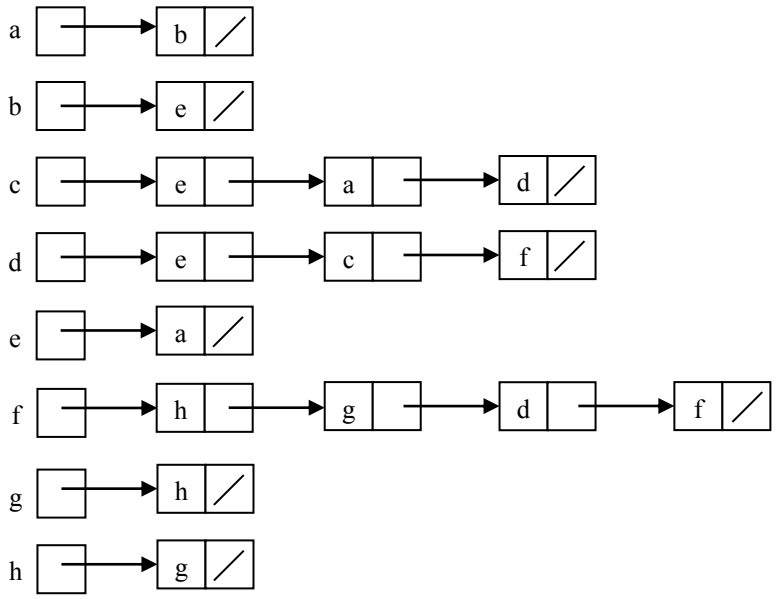
D) Im nächsten Schritt soll die Funktion *elementHinzufuegen()* implementiert werden, welche ein neues Element in der verketteten Liste erzeugt und die Daten darin ablegt. Die Funktion bekommt den Namen, die Matrikelnummer und die Note übergeben. Der Startzeiger *start* der verketteten Liste ist eine globale Variable, welche zu Beginn mit NULL initialisiert ist. Die Funktion soll ein neues Element für die verkettete Liste erzeugen, dieses mit Daten füllen und zur verketteten Liste hinzufügen.
Hinweis: Das Element muss **nicht** am Ende der verketteten Liste hinzugefügt werden.

```
void elementHinzufuegen( string name, short matrikel, float note ) {




}
```

E) Schreiben Sie den Pseudocode zur Funktion *elementSuchen()*. Diese erhält als Übergabeparameter die Matrikelnummer des Studenten und hat als Rückgabewert einen Zeiger auf *Student*. Dieser zeigt auf das gesuchte Element, falls es in der Liste vorhanden ist, ansonsten ist der Inhalt NULL.

Aufgabe 5 Graphen und Tiefensuchalgorithmus

Gegeben ist die folgende Adjazenzliste, welche den Graph Z beschreibt:



Gegeben ist der Pseudocode des Tiefensuchalgorithmus:

```
DFS( Z )
  for alle Knoten u in Z
  do farbe[u] = weiss
     vater[u] = NULL
  zeit = 0
  for alle Knoten u in Z
  do if farbe[u] == weiss
     then DFS-VISIT( u )

DFS-VISIT( u )
  farbe[u] = grau; zeit = zeit + 1
  startTime[u] = zeit
  for alle Knoten v aus Adj[u]
  do if farbe[v] == weiss
     then vater[v] = u
          DFS-VISIT( v )
  farbe[u] = schwarz; zeit = zeit + 1
  endTime[u] = zeit
```

Dabei gilt:

- `Z:` gegebener Graph
- `Adj[u]:` Alle Nachfolger des Knotens `u`

A) Die folgenden Fragen beziehen sich auf die Adjazenzliste (Seite 9)

a) Die Adjazenzliste beschreibt einen (bitte ankreuzen und begründen):
Bitte beachten Sie, dass Sie nur Punkte bei einer korrekten Begründung erhalten.

☐ ungerichteten Graphen

☐ gerichteten Graphen

Begründung:

b) Wieviele Knoten und Kanten hat dieser Graph:

Anzahl Knoten: ☐ Anzahl Kanten: ☐

c) Bestimmen Sie den Grad des Knotens f:

d(f) = ☐

d) Bestimmen Sie die Mächtigkeit der Menge der direkt benachbarten Knoten von f

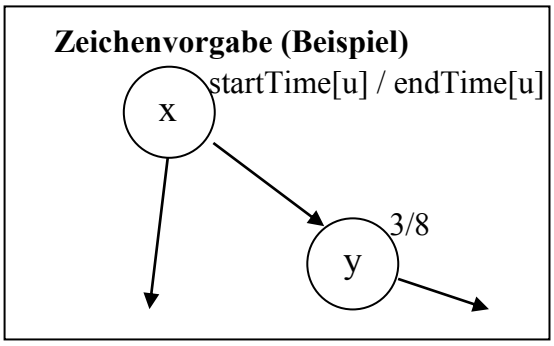
|V'(f)| = ☐

e) Geben Sie eine beliebige einfache Kantenprogression der Länge 4 an:

B) Wenden Sie den Tiefensuchalgorithmus (DFS) auf die Adjazenzliste (Seite 9) an, um festzustellen, ob ausgehend vom Knoten c der Knoten g erreichbar ist.

a) Arbeiten Sie den DFS-Algorithmus vollständig (alle Konten mit Entdeckungs- und Endzeit) ab. Zeichnen Sie den entstehenden Tiefensuchbaum / -wald ausgehend von Startknoten c.

- Die Entdeckungsreihenfolge ist dabei bestimmt durch die Adjazenzliste.



- Bestimmen Sie für jeden Knoten die Entdeckungszeit startTime[u] und die Endzeit endTime[u]

b) Bei welcher Entdeckungszeit könnte der Algorithmus abgebrochen werden, weil der Zielknoten g entdeckt wurde?

möglicher Algorithmus Abbruch bei: ☐

Aufgabe 6 Algorithmus-Analyse: Sortieren

Gegeben ist ein Echtzeitsystem, in dem Zahlen in einem Array sortiert werden. Zum Sortieren wird der einfache, deterministische Algorithmus Bubblesort (deutsch „Blasensortierung“) angewendet. Um die Echtzeitbedingung nicht zu verletzten, muss die maximale Größe des Arrays begrenzt werden. Dazu ist eine Feinlaufzeitanalyse notwendig.

Der relative Aufwand für verschiedene Aktionen und Operationen ist in untenstehender Tabelle zusammengefasst.

Elementare Aktion/Operation	**Kosten (Rel. Zeitaufwand)**
Zuweisung	1,0
Addition/Subtraktion	1,4
Multiplikation	2,3
Division	8,0
Vergleich (inkl. Sprung bei *if* oder *while*)	1,5
Indizierung einer Matrix/Array	3,2

A) Führen Sie die Feinlaufzeitanalyse für den **Worst Case Fall** auf der nächsten Seite entsprechend aus.

Hinweis: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Schreiben Sie in der Spalte „Anzahl von Ausführungen“ den Ausdruck für die Anzahl von Iterationen der entsprechenden Zeile abhängig von der Arraygröße ***n***.

Geben Sie in der Spalte „Kosten“ die Gesamtsumme der Kosten für die Aktion/Operation der entsprechenden Zeile an. Achten Sie darauf, dass in einer Zeile mehrere elementare Aktionen und Operationen möglich sind.

Worst case Betrachtung

Zeile		Anzahl von Ausführungen	Kosten
--------	`int i, j, n; int A[n];` `// weitere Zeilen`	----------	----------
1	`i = 0;`		
2	`while ( i < n - 1 ) {`		
3	`  j = 1;`		
4	`  while ( j < n - i ) {`		
5	`    if ( A[j-1] > A[j] ) {`		
6	`      temp = A[j-1];`		
7	`      A[j-1] = A[j];`		
8	`      A[j] = temp;` `    }`		
9	`    ++j;` `  }`		
10	`  ++i;` `}`		

Aufgabe 7 Binärbaum

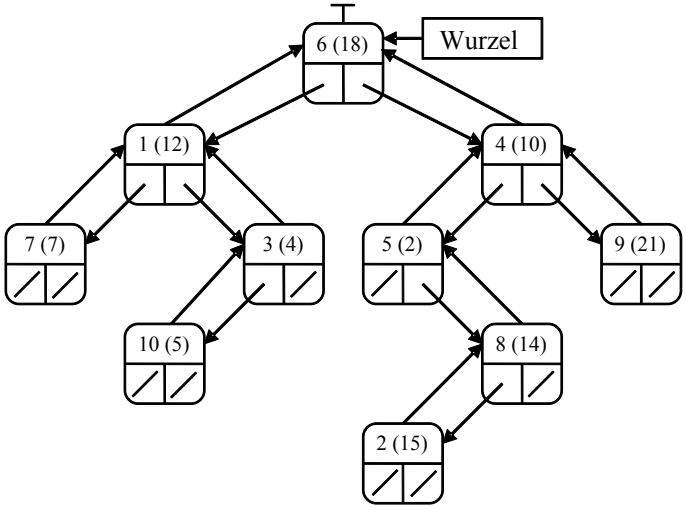
A) Schreiben Sie einen Algorithmus in Pseudocode, der für einen beliebigen Binärbaum mit *n* Knoten die Schlüssel aller Knoten ausgibt. Die Reihenfolge ist dabei unwichtig.

Der Pseudocode wird dabei mit Übergabe des Wurzelelements

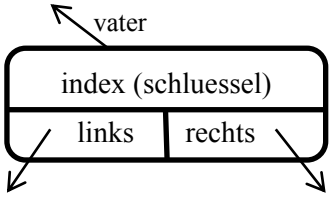
```
Schluesselausgabe( Wurzelelement );
```

aufgerufen.

Als Beispiel ist der folgende Binärbaum gegeben und eine Beschreibung der Elemente jedes Konten als C++ Klassendeklaration. Bitte verwenden Sie bei der Beschreibung des Pseudocodes die entsprechenden Attribute (es werden je nach Algorithmus nicht alle Attribute zur Lösung der Aufgabenstellung benötigt).




```
class Knoten {
  public:
    int index;
    int schluessel;
    Knoten* vater;
    Knoten* links;
    Knoten* rechts;
};
```



Hinweis: Das Problem lässt sich leichter durch einen rekursiven Ansatz lösen.