Prof. Dr.-Ing. K. D. Müller-Glaser

Institut für Technik der Informationsverarbeitung, KIT

Informationstechnik

SS 2010

Probeklausur SS 2010

Mi., 14.7.2010

Hinweise zur Klausur

Hilfsmittel

Zur Prüfung sind keine Hilfsmittel zugelassen. Nicht erlaubt sind insbesondere die Verwendung eines Mobiltelefons und jegliche Kommunikation mit anderen Personen.

Prüfungsdauer

Die Prüfungsdauer beträgt 50 Minuten.

Prüfungsunterlagen

Die Prüfungsunterlagen bestehen aus insgesamt 14 Seiten Aufgabenblättern (einschließlich diesem Titelblatt und zusätzlicher Lösungsblätter).

Bitte vermerken Sie vor der Bearbeitung der Aufgaben auf jeder Seite oben Ihren Namen, auf der ersten Seite zusätzlich die Matrikelnummer!

Falls Sie zusätzliche Blätter zur Lösung der Aufgaben benötigen, verwenden Sie die zusätzlichen Seiten am Ende der Klausur bzw. fragen Sie nach zusätzlichem Papier. Auf jedes zusätzliche Lösungsblatt ist neben dem Namen auch die Aufgaben- und die Seitennummer mit einzutragen. Vermeiden Sie das Beschreiben der Rückseiten. Die Verwendung von eigenen Blättern ist nicht erlaubt. Am Ende der Prüfung sind die 14 Seiten Aufgaben- und Lösungsblätter und alle verwendeten zusätzlichen Lösungsblätter 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)

Aufgabe	1	2	3	4	5	6			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)

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

Aufgabe 2 C++ Verständnisfragen

Beantworten Sie folgende Fragen:

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) Nach Ausführung der folgenden Anweisungen, welchen Wert speichert der String *s*?

```
string s( "IN EZ" );
s += "!";
int pos = s.find( " " );
s.insert( pos + 1, "FO RUL" );
s.erase( pos, 1 );
```

s = ☐

C) Nach Ausführung der folgenden Schleife, welchen Wert speichert die Variable *x*?

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

x = ☐

D) Was ist das Ergebnis des folgenden C++ Ausdrucks? Dabei ist die Variable *Z* vom Typ *bool*.

a) `Z = !((10 < 20) ^ (5 % 3))` ☐ *true* ☐ *false*

b) `Z = ((5 < 10) - (3 & 1)) == 0;` ☐ *true* ☐ *false*

E) 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:

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

Antwort:

Aufgabe 3 Datenstrukturen

A) Nennen Sie mindestens 5 verschiedene Datenstrukturen

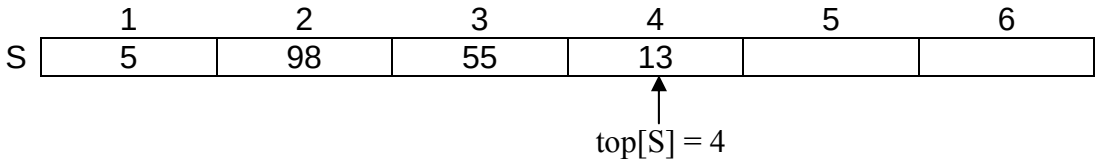
B) Ein Array wird stets in einem zusammenhängenden Speicherbereich abgelegt

☐ Richtig ☐ Falsch

C) Die folgenden Zeilen Programmcode sollen das *float* Array *zahlen* mit insgesamt 12 Werten anlegen und dieses in einer For-Scheife mit Werten von 1,00 bis inkl. 1,11 (Abstand: 0,01) füllen.

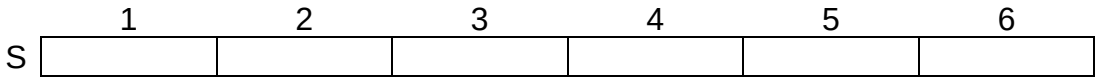
D) Zeichnen Sie ein Nassi-Shneiderman Diagramm, welches einen Algorithmus beschreibt um die Fakultät einer Zahl zu berechnen. Die Zahl soll dabei von der Tastatur eingelesen werden und vor der Berechnung überprüft werden, ob Sie gültig (größer oder gleich Null) ist. Bei einer negativen Zahl soll eine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

E) Gegeben Sei der folgende Stack. Bitte geben die entsprechende Ausgabe zu den darauffolgenden Zeilen an.




```
S.push( 99 );
System.out.println( S.pop() );
System.out.println( S.pop() );
System.out.println( S.peek() );
S.push( 78 );
System.out.println( S.pop() );
System.out.println( S.pop() );
System.out.println( S.peek() );
```

Geben Sie (nach Ausführung) den Inhalt des Stacks und den Stackpointer an.



Aufgabe 4 Binärbaum

A) Zeichnen Sie den binären Baum, der durch die Attribute

Index	Schlüssel	Links	Rechts
1	12	7	3
2	15	NIL	NIL
3	4	10	NIL
4	10	5	9
5	2	NIL	8
6	18	1	4
7	7	NIL	NIL
8	14	2	NIL
9	21	NIL	NIL
10	5	NIL	NIL

dargestellt wird. Knoten 6 sei die Wurzel des binären Baums.

B) Schreiben Sie einen Algorithmus in Pseudocode, der für einen gegebenen Binärbaum mit *n* Knoten die Schlüssel aller Knoten ausgibt.

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

Aufgabe 5 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 5.1 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.

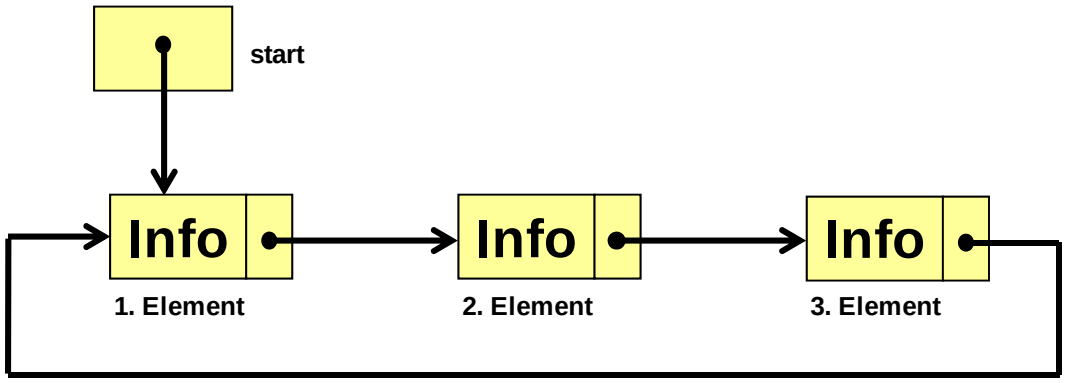


Abbildung 5.1 prinzipieller Aufbau der verketteten Liste

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.

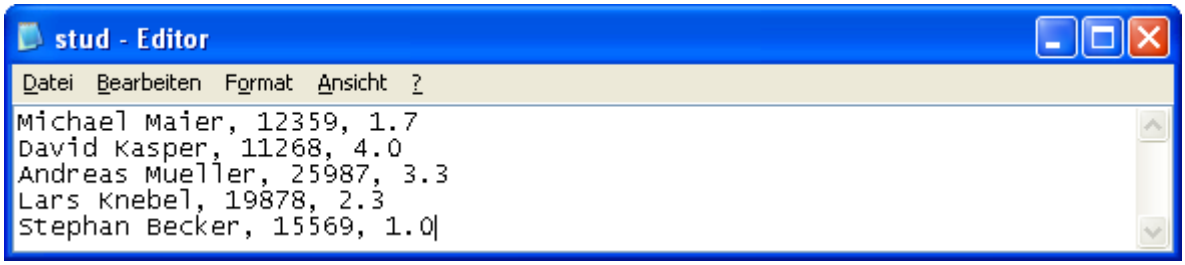


Abbildung 5.2 prinzipieller Aufbau der Datei

A) Geben Sie den Programmcode an, um die oben stehende Datei *stud.txt* zum Lesen zu öffnen. Und geben Sie zwei 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 der die Datei Zeilenweise bis zum Dateiende ausgelesen wird. Bitte geben Sie nur den Schleifenkopf an. Eventuell verwendete Variablen müssen vorher deklariert werden.

C) Es sei eine Zeile aus der Datei ausgelesen und der Funktion *Zeile_auswerten()* übergeben. Ergänzen Sie diese 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 Variablen zuweisen. Es kann davon ausgegangen werden, dass alle Zeilen das gleiche Format haben und immer alle Werte vorhanden sind, wie in Abbildung 5.2 gezeigt. Die String-Variablen werden anschließend, wenn nötig, umgewandelt und dann der Funktion *Element_hinzufuegen()* übergeben – siehe Aufgabenteil D).

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

```
  short matrikel = matrikel_umwandeln( s_matrikel );
  float note = note_umwandeln( s_note );

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

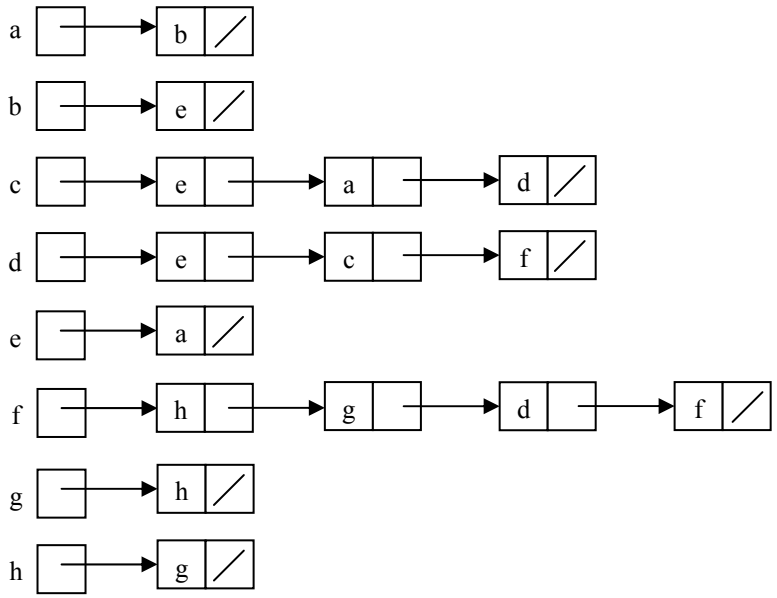
D) Im nächsten Schritt soll die Funktion *Element_hinzufuegen()* implementiert werden, welche ein neues Element in der verketteten Liste erzeugt und die Daten darin ablegt. Die Funktion bekommt den Namen, die Matrikelnummer und die Note übergeben. Der Startzeiger *start* der verketteten Liste ist eine 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 Element_hinzufuegen( string name, short matrikel, float note ) {




}
```

E) Im Programm soll zusätzlich die Funktion *Element_suchen()* implementiert werden. 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. Schreiben Sie zu dieser Funktion den entsprechenden Pseudocode.

Aufgabe 6 Graphen und Tiefensuchalgorithmus

Gegeben ist die folgende Adjazenzliste:



Gegeben ist der Pseudocode des Tiefensuchalgorithmus:

```
DFS( G )
  for alle Knoten u in G
  do farbe[u] = weiss
     vater[u] = NIL
  zeit = 0
  for alle Knoten u in G
  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
```

A) Die folgenden Fragen beziehen sich auf die Adjazenzliste (Seite 12)

a) Die Adjazenzliste beschreibt einen (bitte ankreuzen und begründen):

☐ 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) = ☐

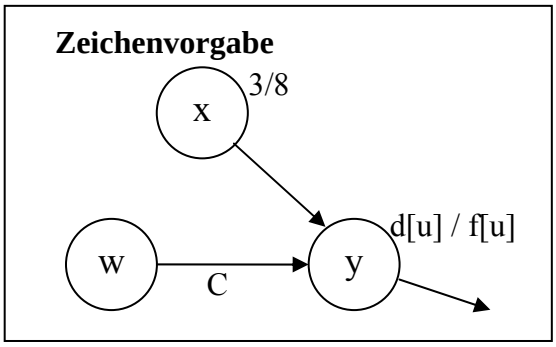
d) Bestimmen Sie die Mächtigkeit der Menge der direkt benachbarten Knoten von f

|V'(f)| = ☐

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

B) Wenden Sie den Tiefensuchalgorithmus (DFS) auf die Adjazenzliste (Seite 12) an, um festzustellen, ob ausgehend vom Knoten c der Knoten g erreichbar ist.

a) Arbeiten Sie den DFS-Algorithmus vollständig ab. Zeichnen Sie den entstehenden Tiefensuchbaum ausgehend von Startknoten c.



- Die Entdeckungsreihenfolge ist dabei bestimmt durch die Adjazenzliste.
- Bestimmen Sie für jeden Knoten die Entdeckungszeit d[u] und die Endzeit f[u]
- Ergänzen Sie alle restlichen Kanten und klassifizieren Sie die Kanten und markieren Sie

  Baumkanten: keine Markierung
  Vorwärtskanten: mit einem F (Forward)
  Rückwärtskanten: mit einem B (Backward)
  Querkanten: mit einem C (Cross)

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

möglicher Algorithmus Abbruch bei: ☐