Dipl.Phys. Gerald Kempfer

Beuth Hochschule für Technik Berlin
- University of Applied Sciences -
Fachbereich VI – Informatik und Medien
Studiengang Technische Informatik Bachelor

Klausur zur Lehrveranstaltung
AKT7 TI-B (Ausgewählte Kapitel der Technischen Informatik)
16. Juli 2010

Hinweise:

Die Teilnahme an der Klausur ist nur bei Bestehen der Übungsaufgaben zulässig!

Zum Bestehen der Klausur benötigen Sie 50 Punkte von 100 möglichen Punkten.

Die Bearbeitungszeit der Klausur beginnt pünktlich um 14.15 Uhr und endet – ebenfalls pünktlich – um 15.45 Uhr. Sie haben also 90 Minuten Zeit.

Folgende Hilfsmittel in Papierform sind zugelassen: Skripte, Mitschriften und Bücher. **Nicht** zugelassen sind elektronische Geräte (Handys, Notebooks, PDAs, usw.).

Schreiben Sie mit Kugelschreiber oder Füller, **nicht** mit Bleistift! Verwenden Sie **nicht** die Farbe Rot. Schreiben Sie leserlich – was ich nicht lesen kann, ist grundsätzlich falsch! Beschreiben Sie nur die Vorderseiten!

Jeder Austausch mit anderen Personen wird als Täuschungsversuch gewertet und führt dazu, dass die Klausuren aller Beteiligten als „nicht bestanden" gewertet werden.

Erläuterungen sollten kurz, aber dennoch präzise und vollständig sein. Wenn möglich ist eine stichpunktartige Beantwortung zu wählen, sofern die Verständlichkeit gegeben ist. Im Zweifelsfall können ganze Sätze Klarheit schaffen.

Kennzeichnen Sie jedes Blatt, das Sie abgeben, mit Ihrem Namen und / oder Ihrer Matrikelnummer.

Viel Erfolg!

Name: [redacted]

Matrikelnummer: [redacted] letzter Prüfungsversuch: ☐

Bewertung:

Aufgabe	Mögliche Punktzahl	Erreichte Punktzahl
1	15	13
2	25	24
3	15	13
4	15	11
5	30	~~25~~ 24
Summe	**100**	85

Note Klausur 1,7

Aufgabe 1: Multiple-Choice-Fragen. (13 / 15)
Kreuzen Sie an, wie der Satz richtig heißen muss. Es gibt immer nur eine richtige Antwort.

Der Referenzoperator bei einer Initialisierung
- ☐ kann links und rechts vom Gleichheitszeichen stehen.
- ☒ steht immer links vom Gleichheitszeichen.
- ☐ steht immer rechts vom Gleichheitszeichen.

```
class Z { };
Z z;
```
- ☐ beinhaltet einen Syntaxfehler:
- ☒ legt das Objekt `z` der Klasse `Z` an.
- ☐ legt das Objekt `Z` der Klasse `z` an.

Der `this`-Zeiger
- ☒ wird automatisch definiert.
- ☐ kann in allen Methoden verwendet werden.
- ☐ kann nur in `public`-Methoden verwendet werden.

Eine Funktion ist überladen, wenn
- ☐ wenn sie im Programm mehrmals aufgerufen wird.
- ☐ sie übergewichtig ist.
- ☒ wenn es weitere Funktionen mit dem gleichen Namen aber unterschiedlichen Parametern und/oder Rückgabedatentypen gibt.

Der Konstruktor der Basisklasse wird beim Erzeugen eines Objektes einer Unterklasse
- ☒ vor dem Konstruktor der Unterklasse aufgerufen.
- ☐ nach dem Konstruktor der Unterklasse aufgerufen.
- ☐ nicht aufgerufen.

Eine `friend`-Funktion darf
- ☐ auf private Elemente der befreundeten Klasse nur lesend zugreifen.
- ☐ nicht auf `protected`-Elemente der befreundeten Klasse zugreifen.
- ☒ auf alle Elemente der befreundeten Klasse zugreifen.

Templates werden auch
- ☐ genetische Typen genannt.
- ☒ Schablonen genannt.
- ☐ temporäre Typen genannt.

Wird bei der Vererbung einer Struktur (`struct`) kein Zugriffsattribut angegeben, wird automatisch das Zugriffsattribut
- ☒ `private` verwendet.
- ☐ `protected` verwendet.
- ☒ `public` verwendet.

Der Standard-Konstruktor hat
- ☐ genauso viele Parameter wie der Kopier-Konstruktor.
- ☒ einen Parameter mehr als der Kopier-Konstrutktor.
- ☒ einen Parameter weniger als der Kopier-Konstruktor.

`::` ist ein
- ☒ Bereichsoperator.
- ☐ Beendigungsoperator.
- ☐ Begleichungsoperator.

Eine Referenz ist
- ☒ ein Aliasname.
- ☐ eine Adresse.
- ☐ ein Zeiger.

`flush` ist
- ☐ eine Konstante.
- ☒ ein Manipulator.
- ☐ eine Systemvariable.

Der Datentyp für Wahrheitswerte (benannte nach George Boole) heißt
- ☒ `bool`.
- ☐ `boolean`.
- ☐ `boole`.

Der `this`-Zeiger zeigt auf das Objekt,
- ☐ das zuerst erzeugt wurde.
- ☐ das zuletzt erzeugt wurde.
- ☒ dessen Methode aufgerufen wurde.

Der Konstruktor einer Klasse
- ☒ hat den gleichen Namen wie die Klasse.
- ☐ wird grundsätzlich in Kleinbuchstaben geschrieben.
- ☐ beginnt mit einer Tilde (~).

Aufgabe 2: **a)** Was gibt das folgende Programm aus? Schreiben Sie die Ausgaben auf das nächste Blatt! (20 / 20)

```
#include <iostream>

using namespace std;

class THimmelskoerper
{
   protected:
      double Dichte;
   public:
      THimmelskoerper()          { cout << "+ HK" << endl; }
      virtual ~THimmelskoerper() { cout << "- HK" << endl; }
};

class TFesterHimmelskoerper: public THimmelskoerper
{
   public:
      double Durchmesser;
      TFesterHimmelskoerper()          { cout << "+ FHK" << endl; }
      virtual ~TFesterHimmelskoerper() { cout << "- FHK" << endl; }
};

class TPlanet: public TFesterHimmelskoerper
{
   public:
      int AnzahlKontinente;
      TPlanet()          { cout << "+ P" << endl; }
      virtual ~TPlanet() { cout << "- P" << endl; }
};

class TBewohnterPlanet: public TPlanet
{
      long int AnzahlBewohner;
   public:
      TBewohnterPlanet()          { cout << "+ BP" << endl; }
      virtual ~TBewohnterPlanet() { cout << "- BP" << endl; }
};

int main()
{
   THimmelskoerper *Himmelskoerper;  1.
   TPlanet *Mars;  2.
   TBewohnterPlanet Erde;  3.

   cout << "Beginn Hauptprogramm" << endl;  4.
   Himmelskoerper = new TFesterHimmelskoerper;  5.
   Mars = new TPlanet;  6.
   // Zeilen für zweiten Teil der Aufgabe:
   // Himmelskoerper->Dichte = 4.219;
   // Himmelskoerper->AnzahlMonde = 4;
   // Mars.AnzahlKontinente = 1;
   // Erde.AnzahlKontinente = 7;
   // Erde.AnzahlBewohner = 5938784294L;
   // Ende Zeilen für zweiten Teil der Aufgabe
   delete Himmelskoerper;  7.
   delete Mars;  8.
   cout << "Ende Hauptprogramm" << endl;  9.

   return 0;
}
```

Ausgabe des oben stehenden Programms:

+HK
+FHK } 3.
+P
+BP

Beginn Hauptprogramm } – 4

+HK } 5
+FHK
+HK
+FHK } 6
+P
–FHK } 7
–HK
–P
–FHK } 8
–HK
Ende Hauptprogramm } 9
–BP
–P
–FHK
–HK

Erde wird bei Beendigung zerstört (vernichtet)

✓

b) Wenn die folgenden Zeilen, die in dem oben stehenden Hauptprogramm auskommentiert sind, eingefügt werden würden, welche Zeilen sind korrekt und welche sind nicht korrekt? (4 / 5)

`Himmelskoerper->Dichte = 4.219;`	// ☐ korrekt	☒ nicht korrekt	✓
`Himmelskoerper->AnzahlMonde = 4;`	// ☐ korrekt	☒ nicht korrekt	✓
`Mars.AnzahlKontinente = 1;`	// ☒ korrekt	☒ nicht korrekt	✓
`Erde.AnzahlKontinente = 7;`	// ☒ korrekt	☐ nicht korrekt	✓
`Erde.AnzahlBewohner = 5938784294L;`	// ☒ korrekt	☒ nicht korrekt	✓ private

Aufgabe 3: Gegeben sind vier Klassen sowie einige Eigenschaften:

Klassen	**Eigenschaften**
Rechteck	Anzahl Ecken (`int`)
Vieleck	Kantenlänge A (`float`)
GeometrischeForm	Kantenlänge B (`float`)
Quadrat	Linienbreite (in Pixel; `int`)
	Linienfarbe (Farbnummer; `int`)
	Füllfarbe (Farbnummer; `int`)

Erstellen Sie aus diesen Klassen eine Klassenhierarchie: Welche Klasse ist die Basisklasse und welche werden in welcher Reihenfolge abgeleitet? Definieren Sie dazu im unten stehenden Quelltext passend zum Hauptprogramm die Klassen mit den privaten Eigenschaften und mit jeweils einem Konstruktor, der die Eigenschaft setzt. (13 / 15)

```
#include <iostream>

using namespace std;

// Platz für die vier Klassen:
```

class GeometrischeForm
{
int Linienbreite;
int Linienfarbe;
int Fuellfarbe;
public:
GeometrischeForm(int LB, int LF, int FB):
Linienbreite(LB), Linienfarbe(LF), Fuellfarbe(FB) {};
};

Da eine Linie auch eine geometrische Form ist, gehört die Füllfarbe eigentlich zum Vieleck.

class Vieleck : public GeometrischeForm
{
int AnzEcken;
public:
Vieleck(int AE, int LB, int LF, int FB):
GeometrischeForm(LB, LF, FB),
AnzEcken(AE) {};
};

class Rechteck: public Vieleck
{
float KantenlaengeA;
float KantenlaengeB;
public:
Rechteck(float KA, float KB, int LB, int LF, int FB):
Vieleck(4, LB, LF, FB),
KantenlaengeA(KA),
KantenlaengeB(KB) {};
};
weiter bei (2) >

class Quadrat: public Rechteck
{
float KantenlaengeA; — überflüssig
public:
Quadrat(float KA, int LB, int LF, int FB):
Vieleck(4, LB, LF, FB), KantenlaengeA(KA)
{}; Rechteck(KA, KA, LB, LF, FB) {}
};

-2

```
int main()
{
    Quadrat Q(3.5, // Kantenlaenge
              1,   // Fuellfarbe Nr. 1 (weiss)
              0,   // Linienfarbe Nr. 0 (schwarz)
              2 ); // Linienbreite in Pixel
}
```

Aufgabe 4: Definieren Sie eine Template-Funktion namens `tausche`, die zwei Referenz-Parameter vom Typ des Template-Parameters erhält. Die Template-Funktion soll die Werte der beiden Parameter austauschen.

(11 / 15)

```
#include <iostream>

using namespace std;

// Platz für Ihre Templatefunktion (Deklaration und Definition):
```

template <class T> T tausche (int Zahl1, int Zahl2)
{
int zwischen = Zahl1;
Zahl1 = Zahl2;
Zahl2 = zwischen
(*)

template <class T> T tausche (T Erstes, T Zweites)
{
return (Zweites, Erstes);
}

Es sind Referenz-Parameter gefordert!
-1

Damit wird nichts getauscht!
T Temp = Erstes;
Erstes = Zweites;
Zweites = Temp;
-3

siehe auch (*)
Dann ist auch der Rückgabewert überflüssig!

```
int main()
{
   int Zahl1 = 3, Zahl2 = 5;
   string Text1 = "Hallo", Text2 = "Welt";

   cout << "Vor Tauschen:" << endl;
   cout << "Zahl1 = " << Zahl1 << endl;   // 3
   cout << "Zahl2 = " << Zahl2 << endl;   // 5
   cout << "Text1 = " << Text1 << endl;   // Hallo
   cout << "Text2 = " << Text2 << endl;   // Welt

   tausche(Zahl1, Zahl2);
   tausche(Text1, Text2);

   cout << "Nach Tauschen:" << endl;
   cout << "Zahl1 = " << Zahl1 << endl;   // 5
   cout << "Zahl2 = " << Zahl2 << endl;   // 3
   cout << "Text1 = " << Text1 << endl;   // Welt
   cout << "Text2 = " << Text2 << endl;   // Hallo
}
```

Aufgabe 5: In der Schule haben Sie bestimmt gelernt, dass man Tonnen und Kisten nicht addieren kann. Hier der Gegenbeweis:
In den folgenden Klassen fehlen noch Methoden bzw. Funktionen, damit das Hauptprogramm korrekt laufen kann. Ergänzen Sie die gegebenen Klassen um die nötigen Operatoren, um
- Tonnen zu addieren,
- Kisten zu addieren sowie
- Tonnen und Kisten zu addieren.

Es werden jeweils Gewicht und Volumen addiert. Dabei ergeben Tonnen + Kisten ein Objekt der Klasse Tonne und Kisten + Tonnen ein Objekt der Klasse Kiste.
Ferner muss noch der Ausgabeoperator überladen werden. Dabei sollen Gewicht und Volumen der Tonnen, Kisten und Tonnen-Kisten-Gemische ausgegeben werden (siehe auch Ausgabe des Programmes).
Beachten Sie dabei die Namensräume! (24 / 30)

Ausgabe des Programmes:

```
Tonnen:
  Gewicht 1750 kg
  Volumen 350 m³

Kisten:
  Gewicht 1250 kg
  Volumen 260 m³

Tonnen und Kisten:
  Gewicht 1550 kg
  Volumen 290 m³

Kisten und Tonnen:
  Gewicht 1450 kg
  Volumen 320 m³
```

Vorgegebenes Programm:

```
#include <iostream>

using namespace std;

namespace Kisten
{
   class Kiste; // Vorwärtsdeklaration
}

namespace Tonnen
{
   class Tonne
   {
         double Gewicht;
         double Volumen;
         bool Gemischt;  // Tonnen und Kisten gemischt
      public:
         Tonne(double G = 0.0, double V = 0.0, bool Gem = false):
               Gewicht(G), Volumen(V), Gemischt(Gem) { }
         double getGewicht() { return Gewicht; }
         double getVolumen() { return Volumen; }
```

```
            // Platz für Ihre Deklarationen (3 Punkte):
```

Tonne operator + (Tonne &t);
Tonne operator + (Kisten::Kiste &k);
friend ostream &operator << (ostream &ostr, Tonne &t);

```
    };
} // Namespace Tonnen

namespace Kisten
{
    class Kiste
    {
          double Gewicht;
          double Volumen;
          bool Gemischt;  // Kisten und Tonnen gemischt
       public:
          Kiste(double G = 0.0, double V = 0.0, bool Gem = false):
                Gewicht(G), Volumen(V), Gemischt(Gem) { }
          double getGewicht() { return Gewicht; }
          double getVolumen() { return Volumen; }
          // Platz für Ihre Deklarationen (3 Punkte):
```

Kiste operator + (Kiste &k);
Kiste operator + (Tonnen::Tonne &t);
friend ostream &operator << (ostream &ostr, Kiste &k);

```
    };
} // Namespace Kisten

int main()
{
   Tonnen::Tonne T1(800, 150), T2(950, 200), T3;
   Kisten::Kiste K1(500, 120), K2(750, 140), K3;

   T3 = T1 + T2;
   K3 = K1 + K2;
   cout << T3 << endl;
   cout << K3 << endl;

   T3 = T1 + K2;
   K3 = K1 + T2;
   cout << T3 << endl;
   cout << K3 << endl;

   return 0;
} // main

// Platz für Ihre Definitionen (24 Punkte) auf der nächsten Seite:
```

Fortsetzung der Definitionen für Aufgabe 5: Die Definitionen müssen im Namespace stehen! Das using reicht nicht! (−2)

```
using namespace Tonnen;
{
Tonne Tonne::operator+(Tonne &t)
 {
  return Tonne(Gewicht + t.getGewicht(), Volumen + t.getVolumen());
 }

Tonne Tonne::operator+(Kisten::Kiste &k)
{
  return Tonne(Gewicht + k.getGewicht(), Volumen + k.getVolumen(), true);
ostream &operator<<(ostream &ostr, Tonne &t)
 { if (t.Gemischt)
     ostr << "Tonnen und Kisten" << endl;
   else
     ostr << "Tonnen" << endl;
   ostr << "Gewicht=" << t.Gewicht << "kg" << "Volumen=" << t.Volumen << "m3" << endl;
   return ostr;
 }
}
using namespace Kisten;
{
Kiste Kiste::operator+(Kiste &k)
{
 return Kiste(Gewicht + k.getGewicht(), Volumen + k.getVolumen());
}
Kiste Kiste::operator+(Tonnen::Tonne &t)
{
 return Kiste(Gewicht + t.getGewicht(), Volumen + t.getVolumen(), true);
ostream &operator<<(ostream &ostr, Kiste &k)
 { if (k.Gemischt)
     ostr << "Kisten und Tonnen" << endl;
   else
     ostr << "Kisten" << endl;
   ostr << "Gewicht" << k.Gewicht << "kg" << "Volumen=" << k.Volumen << "m3" << endl;
   return ostr;
 }
}
```

(−1) (−2) endl<< wegen vorgegebener Ausgabe (−1) (✓) (✓) s.o.