Name:

Dr.-Ing. Hartmut Helmke
Fachhochschule
Braunschweig/Wolfenbüttel
Fachbereich Informatik

Matrikelnummer:	*Punktzahl:*
Ergebnis:	

Freiversuch ☐ *F1* ☐ *F2* ☐ *F3* ☐

Klausur im WS 2006/07 :

☐ **Programmierkonzepte — Lösungen —**
☐ **Informatik III — Lösungen —**

Medieninformatik ☐ *Praktische Informatik* ☐ *Technische Informatik* ☐
Informatik B. Sc. ☐ *Technische Informatik B. Sc.* ☐

Hilfsmittel sind bis auf Computer, Handy etc. erlaubt !	Bitte Aufgabenblätter mit abgeben !
Austausch von Hilfsmitteln mit Kommilitonen ist **nicht** erlaubt !	

Die Lösungen können in einigen Fällen hier auf dem Aufgabenblatt angegeben werden. Sie dürfen aber auch Ihre Lösungen, falls erforderlich, auf separaten Blättern notieren.
Bitte notieren Sie auf **allen** Aufgabenblättern und separaten Blättern Ihren Namen bzw. Ihre Matrikelnummer.
Auf eine absolut korrekte Anzahl der Blanks und Zeilenumbrüche braucht bei der Ausgabe nicht geachtet zu werden. Dafür werden keine Punkte abgezogen.
Hinweis: In den folgenden Programmen wird manchmal die globale Variable *datei* verwendet. Hierfür kann der Einfachheit halber die Variable *cout* angenommen werden. Die Variable *datei* diente lediglich bei der Klausurerstellung dem Zweck der Ausgabeumlenkung.

Geplante Punktevergabe

Punktziel	Im einzelnen	Pkte
Hausaufgaben: 10 P.		
A1: 12P.	(3+2+2+3+2)	
A2: 30P.	(2+8+1+4+2+2+2+5+4)	
A3: 15P.	(1+3+2+2+1+1+1+4)	
A4: 15P.	(2+2+2+3+1+5)	
A5: 28P.	(9+5+5+9)	
Summe 100 P.		

Aufgabe 1 : Klassen

ca. 12 Punkte
Gegeben ist die folgende Deklaration der Klasse `Fliese`.

```
// quadratische Fliese
class Fliese {
private:
   double breite; // in m
public:
   double getBreite() const;
   void setBreite(double b);

   Fliese(double b) : breite(b)
      {datei << "+F" << b << " ";}
   ~Fliese() {datei << "-F" << breite << " ";}

   // Kopierkonstruktor
   // Zuweisungsoperator;
   };
```

a.) Implementieren Sie zunächst die beiden Methoden `getBreite` und `setBreite`. **Lösung:**

```
void Fliese::setBreite(double b) {
    breite = b;
   }

double Fliese::getBreite() const {
   return breite;
   }
```

b.) Was bedeutet das Schlüsselwort `const` bei der Methode `getBreite`? Warum fehlt es bei der Methode `setBreite`? **Lösung:**
`getBreite` verändert keine Attribute des Objektes, `setBreite` dagegen schon.

c.) Geben Sie einen Include-Wächter für die Header-Datei an.

```
#   ifndef FlieseTapete
#   define FlieseTapete
. . .
#endif
```

d.) Implementieren (cpp-Datei) Sie nun den Kopierkonstruktor und Zuweisungsoperator der Klasse `Fliese`. **Lösung:**

```
Fliese::Fliese(const Fliese& t2) {
   breite = t2.breite;
   }

const Fliese& Fliese::operator=(const Fliese& t2) {
   breite = t2.breite;
   return *this;
   }
```

e.) Warum würde das folgende Codefragment beim Übersetzen einen Syntaxfehler ergeben?

```
Fliese fliesArray[22];
```

Lösung:
Es fehlt der Default-Konstruktor bei der Klasse.

Aufgabe 2 : Klassen, 2. Teil

ca. 30 Punkte
Wir betrachten nochmals die Header-Datei der Klasse `Fliese` mit dem gleichen Inhalt wie zuvor.

```
// quadratische Fliese
class Fliese {
private:
   double breite; // in m
public:
   double getBreite() const;
   void setBreite(double b);

   Fliese(double b) : breite(b)
      {datei << "+F" << b << " ";}
   ~Fliese() {datei << "-F" << breite << " ";}

   // Kopierkonstruktor
   // Zuweisungsoperator;
   };
```

Im Folgenden dürfen Sie Ihre Ausgaben direkt unter den Funktionen oder auf einem Extra-Blatt notieren.

a.) Welche Ausgabe erzeugt der Aufruf von der Funktion `funk1` in die Datei?

```
void funk1() {
   Fliese f1(3);
   Fliese f2(4);
   }
```

Lösung:

```
funk1: +F3 +F4 -F4 -F3
```

b.) Veranschaulichen Sie die Speicherbelegung im Stack- und im Heap-Programmspeicher zu den Zeitpunkten `/* 1 */`, `/* 2 */` **und** `/* 3 */` im folgenden Programmfragment grafisch.

```
void funk2() {
   Fliese f1(3);
   Fliese* pf2 = new Fliese(4);
   Fliese* pf3 = &f1; /* 1 */
   *pf3 = 44.2;
   pf3 = new Fliese(11); /* 2 */
   pf2 = pf3;
   delete pf2;   /* 3 */
   }
```

c.) Beschreiben Sie, was in der Programmzeile

```
   *pf3 = 44.2;
```

geschieht, d.h. warum kann man eine double-Konstante einer `Fliese` zuweisen (1 Zeile Text reicht)?
Lösung:
Die Klasse hat einen Umwandlungskonstruktor, der

eine `double`-Konstante in eine Instanz der Klasse umwandelt. Hierbei wird der Konstruktor auch ausgeführt!

d.) Welche Ausgabe erzeugt der Aufruf von der obigen Funktion `funk2` in die Datei? **Lösung:**

```
funk2: +F3 +F4 +F44.2 -F44.2 +F11 -F11 -F44.2
```

e.) Implementieren Sie die Funktion `verdoppeln`, die die Breite der `quadratischen Fliese` verdoppelt, d.h. der Aufruf der folgenden Funktion `funk3` soll die Ausgabe `6` auf dem Bildschirm ausgeben.

```
void funk3() {
   Fliese f1(3);
   verdoppeln(f1);
   cout << f1.getBreite() << endl;
   }
```

Lösung:

```
void verdoppeln( Fliese& f) {
   f.setBreite (f.getBreite() * 2);
   }
```

Gegeben ist ferner die Unterklasse `Kachel.`

```
class Kachel: public Fliese {
private:
   string farbe;
public:
   Kachel(double b, string f): Fliese(b), farbe(f)
      {datei << "+K" << f << " ";}
   ~Kachel() {datei << "-K" << farbe << " ";}
   };
```

f.) Zeichnen Sie das UML-Klassendiagramm ohne Methoden und Attribute der beiden Klassen.

g.) Welche Ausgabe erzeugt der Aufruf von der Funktion `funk4` in die Datei?

```
void funk4() {
   Kachel f1(3, "grau");
   }
```

Lösung:

```
funk4: +F3 +Kgrau -Kgrau -F3
```

h.) Veranschaulichen Sie die Speicherbelegung im Stack- und im Heap-Programmspeicher zum Zeitpunkt `/* 1 */` im folgenden Programmfragment grafisch.

```
void funk5() {
   Kachel f1(3, "grau");
   Kachel* pk2 = new Kachel(4, "rot");
   Kachel* pk3 = new Kachel(f1); /* 1 */
   datei << "\n  ";
   delete pk2;
   delete pk3;
   }
```

i.) Welche Ausgabe erzeugt der Aufruf von der obigen Funktion `funk5` in die Datei?
Lösung:

```
funk5: +F3 +Kgrau +F4 +Krot
   -Krot -F4 -Kgrau -F3 -Kgrau -F3
```

Aufgabe 3 : Textfragen, Team

ca. 15 Punkte

a.) Warum beginnen viele eine Iteration an einem Dienstag und nicht an einem Montag? **Lösung:** Montag ist ein Scheißtag. Planung ist eine Scheißarbeit. Warum soll auch noch beides zusammengelegt werden?

Man schließt nicht hektisch kurz vor dem Wochenende noch schnell eine Iteration ab.

b.) Das Team hatte sich in der letzten Iteration (Dauer 33 Arbeitstage) Aufgaben im Umfang von 31 idealen Tagen vorgenommen. Erledigt wurden allerdings nur Aufgaben im Umfang von 30 idealen Tagen. Zusätzlich wurden ungeplante Aufgaben im Umfang von 3 idealen Tagen erledigt. Berechnen Sie den Load Factor (Hinweis $\frac{1}{11} = ca.\quad 0,09$)? (Rechenweg angeben)

Die neue Iteration dauert 50 Arbeitstage. Wie viel Aufgaben (Angabe in idealen Tagen) sollte sich das Team vornehmen, wenn der Load Factor der Iteration als Entscheidungsgrundlage verwendet wird? (Rechenweg angeben)

Lösung:
Load Factor : $\frac{30}{33} = 91\%$, damit kann man sich in der neuen Iteration Aufgaben im Umfang von ca. $(50 * \frac{10}{11} =)$ 45 Tagen vornehmen.

c.) Welche Besonderheit gibt es beim sog. *Wasserfallmodell* gegenüber zum Beispiel einem iterativen Vorgehensmodell zur Softwareentwicklung (ein Satz)? **Lösung:**
Die einzelnen Phasen folgenden aufeinander und sie werden nur einmal nicht mehrfach durchlaufen. In Vorgängerphase müssen alle Entscheidungen, die die Nachfolgerphase benötigt, korrekt getroffen worden sein.

d.) Welche andere Basistechnik von eXtreme Programming ermöglicht es, dass man auf ein Design des gesamten Systems zu Beginn der Software-Entwicklung weitgehend verzichten kann? Begründen Sie kurz Ihre Entscheidung. **Lösung:** Durch Refactoring ist auch noch möglich, dass Design im Nachhinein geänderten Anforderungen anzupassen.

e.) Erklären Sie kurz und knapp: Was ist *Refactoring*? **Lösung:**
Verbesserung des Designs, nachdem der Code

geschrieben wurde, ohne dessen Verhalten zu ändern.

f.) Sie entwickeln in einem Team eine Software. Paul erstellt die Klasse A, Paula die Klasse B, ... und Alfons die Klasse Z. Sie haben alle Änderungen und Erweiterungen an diesen Klassen auf ihren lokalen Arbeitsplätzen durchgeführt. Mit welchem Tool / welchen Tools können Sie sicherstellen, dass jederzeit auf allen lokalen Rechnern/Arbeitsplätzen die gleiche aktuelle Version vorhanden ist? **Lösung:**
Ein Versionsverwaltungssystem, z.B. CVS oder subversion

g.) Was ist die Basis-Voraussetzung, damit ein Refactoring erfolgreich durchgeführt werden kann?
Lösung:
ausreichend Tests vorhanden

h.) Welche Vorteile ergeben sich, wenn man die Tests noch vor Beginn der Implementierung der eigentlichen Funktionalität spezifiziert?
Lösung:
Sie tragen zum Verständnis der zu implementierenden Funktionalität bei. Man verfügt sofort über Anwendungsfälle und hat auch sofort Tests, mit denen man anschließend die Korrektheit der Implementierung überprüfen kann.

Aufgabe 4 : Textfragen, C++, Polymorphie

ca. 15 Punkte

a.) Geben Sie ein Beispiel für eine Klasse mit einer **konstanten** Methode an (Klasse, ein Attribut, die Methode mit Implementierung).

Lösung:

```
class X {
  int x;
public:
  int getX() const {return x;}
};
```

b.)
```
int i = 44,  j = 48;
const int* pc = &i;
```

Welche der folgenden Code-Zeilen ist syntaktisch falsch?

```
*pc = 64;   /* 1 */
pc = &j;    /* 2 */
pc = 41;    /* 3 */
```

Lösung:
Dort wo pc hinzeigt, darf der Inhalt über pc nicht verändert werden. Damit ist Zeile 1 falsch. Zeile 3 ist auch falsch, weil pc ein Zeiger ist und 41 ein int.

Lösung:

c.) Beschreiben Sie kurz den Unterschied zwischen öffentlichen und privaten C++-Methoden?

Lösung:
Öffentliche Methoden gehören zur Schnittstelle der Klasse. Hierauf kann von überall zugegriffen werden. Privaten Methoden können nur von innerhalb der Klasse (eigene Methoden) und von Freunden aufgerufen werden.

d.) Geben Sie ein Beispiel für die Deklaration (Definition nicht erforderlich) einer Klasse mit einer minimalen Standard-Schnittstelle.

Lösung:

```
class X {
public:
   X();
   ~X();
   X(const X&);
   const X& operator=(const X&);
};
```

e.) Geben Sie ein Beispiel für eine ganz kurze abstrakte C++-Klasse an.

Lösung:

```
class X {
public:
   virtual ~X() = 0 {;};
};
```

```
class A{
public:
   virtual void eins() {datei << "\neins in A";}
           void zwei() {datei << ", zwei in A";}
  };

class B: public A{
public:
   virtual void eins() {datei << "\neins in B";}
           void zwei() {datei << ", zwei in B";}
  };
```

f.) Welche Ausgabe erzeugt die Ausführung des folgenden Programmfragments?

```
void poly(){
  B b; b.eins(); b.zwei();
  A a; a.eins(); a.zwei();

  A* pa = new B; pa->eins(); pa->zwei();
  A a2 = *pa; a2.eins(); a2.zwei();
  A* pa2 = &a2; pa2->eins(); pa2->zwei();
```

Lösung:

```
eins in B, zwei in B
eins in A, zwei in A
eins in B, zwei in A
eins in A, zwei in A
eins in A, zwei in A
```

Aufgabe 5 : Test und Doku first

ca. 28 Punkte

Notieren Sie Ihre Lösung bitte auf einem Extrablatt.
Sie wollen eine Software-Lösung für eine Fliesenlegerfirma erstellen, damit Sie besser kalkulieren können, wie viele Fliesen in etwa (nicht ganz genau) für das Fliesen eines gegebenen Raumes erforderlich sind. Der Raum ist jeweils durch seine Länge und Breite gegeben. Außerdem ist die Länge der quadratischen Fliesen gegeben, siehe auch nebenstehenden Code zur Veranschaulichung. Gehen Sie nun bei der Entwicklung der Software-Lösung in den folgenden Schritten vor.
Achtung: Für eine andere Reihenfolge gibt es wahrscheinlich **k e i n e** Punkte!!! a.) und b.) dürfen Sie allerdings in beliebiger Reihenfolge durchführen.
Tipp: Zeichnung anfertigen, könnte zur Klärung beitragen.

a.) Definieren Sie drei wirklich **verschiedene** Tests für die Aufgabenstellung, d.h. für den Test der Funktion `berechneFliesenAnzahl` (Es geht hier um die Beschreibung/Auflistung von konkreten Ein- und Ausgaben der Tests – noch nicht um deren Implementierung in C++).

Lösung:
1. Test: Fliese $1m^2$, Raum 2m lang, 3m breit $--> 6$ Fliesen
2. Test: Fliese $1m^2$, Raum 2,2m lang, 3m breit $--> 9$ Fliesen
2. Test: Fliese $1m^2$, Raum 2,2m lang, 3,2m breit $--> 12$ Fliesen

b.) Falls Sie bestimmte Annahmen getroffen haben, machen Sie diese explizit. (Die Aufgabenstellung war bestimmt noch nicht ganz eindeutig.) Als Software-Entwickler müssen Sie Entscheidungen fällen, die Sie schriftlich festhalten sollen, damit Sie diese mit dem Kunden ggf. diskutieren können.

Lösung:
- Die Fugenbreite und Platz für Fussleiten wird ignoriert
- Rest in der Länge und Breite werden weggeworfen
- Es werden nur ganze Fliesenanzahlen ermittelt, d.h. es wird immer aufgerundet.
- Es wird der gesamte Raum gefliest, der auch als rechteckig angenommen wird.

c.) Implementieren Sie **einen** der zuvor definierten Tests *Test1*, *Test2* und *Test3* in C++.
Anmerkung: Auf die Implementierung der Datei *Test.h* dürfen Sie verzichten.

Verwenden Sie zur Testerstellung die Schnittstellen der Klassen `Fliese` und `Raum`. Legen Sie auch die Schnittstelle der zu testenden Funktion `berechneFliesenAnzahl` fest, d.h. die Deklaration Funktion.

```
// quadratische Fliese
class Fliese {
private:
   double breite; // in m
public:
   double getBreite() const;
   void setBreite(double b);

   Fliese(double b) : breite(b)
      {datei << "+F" << b << " ";}
   ~Fliese() {datei << "-F" << breite << " ";}

   // Kopierkonstruktor
   // Zuweisungsoperator;
   };
```

```
class Raum {
private:
   double laenge; // in m
   double breite; // in m
public:
   Raum(double la, double b) : laenge(la), breite(b)
      {datei << "+R" << la << " ";}
   ~Raum() {datei << "-R" << laenge << " ";}
   /* ... */
   };
```

Lösung:

```
int berechneFliesenAnzahl
   (const Fliese& fli, const Raum& ra);

/** 1. Test: Fliese 1 qm, Raum 2m lang, 3m breit
   --> 6 Fliesen */
bool test1() {
   Fliese fli(1.0);
   Raum ra(2, 3);
   if (berechneFliesenAnzahl(fli, ra) == 6) {
      return true;
      }
   else {
      return false;
      }
   }
```

d.) Beschreiben Sie mit Worten, was die Funktion `berechneFliesenAnzahl` leistet, d.h., was die Eingangs- und Ausgangsvariablen sind und **wie** die Funktion aus den Eingangsparametern die Ausgangsparameter ermittelt, **nicht** implementieren.

Lösung:
Die Funktion ermittelt wie viele Fliesen zur kompletten Überdeckung des Raums benötigt werden. Der Funktion werden die Maße der quadratischen Fliese übergeben. Außerdem wird die Raumgröße als Breite und Länge übergeben.

Die Funktion ermittelt nun zunächst, wie viele Fliesen zur Abdeckung einer kompletten Länge

benötigt werden. Ein evtl. Fliesenrest wird weggeworfen.
- AnzahlLänge = Aufrunden (Raumlänge durch Fliesengröße)

Dann wird ermittelt, wie viele Fliesen zur Abdeckung einer kompletten Breite benötigt werden. Ein evtl. Fliesenrest wird weggeworfen.
- AnzahlBreite = Aufrunden (Raumbreite durch Fliesengröße)

Die gesuchte Fliesenanzahl ist das Produkt aus beiden.
- Anzahl = AnzahlBreite * AnzahlLänge