

Name:

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

Matrikelnummer:		Punktzahl:	
Ergebnis:			
Freiversuch	<input type="checkbox"/>	F1	<input type="checkbox"/>
		F2	<input type="checkbox"/>
		F3	<input type="checkbox"/>

Klausur im WS 2007/08 :

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

Medieninformatik   
Informatik B. Sc.

Praktische Informatik

Technische Informatik   
Technische Informatik B. Sc.

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

Die Lösungen können größtenteils 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

Planen Sie pro Punkt etwas mehr als eine Minute Aufwand ein.

Punktziel	Im einzelnen	Pkte
Übungsaufgaben: 10 P.		
A1: 16 <small>(2+3+3+8)</small> P.		
A2: 18 <small>(1+1+5+1+1+2+2+4+1)</small> P.		
A3: 20 <small>(2+2+1+3+3+5+4)</small> P.		
A4: 11 <small>(1,5+1,5+6+2)</small> P.		
A5: 19 <small>(2+1+1+6+2+2+1+2+2)</small> P.		
A6: 16 <small>(1+2+1+2+1,5+1+1+2+2+2,5)</small> P.		
Summe 100 P.		

**Aufgabe 1 : Stack-Heapspeicher**

ca. 16 (2+3+3+8) Punkte Teil d: 2+2+3+1=8

a.) Veranschaulichen Sie grafisch die Stack-Speicherbelegung des folgenden Programmfragments am Funktionsende.

```
void f1() {
    double d = 12.4;
    int i = 41;
}
```

**Lösung:**

Zu beachten, dass double 8 Byte belegt.

b.) Veranschaulichen Sie grafisch die Stack-Speicherbelegung des folgenden Programmfragments am Funktionsende.

```
void f3() {
    int i = 41;
    int* pi = &i;
    int* p2 = pi;
}
```

c.) Veranschaulichen Sie grafisch die Stack- **und** Heap-Speicherbelegung des folgenden Programmfragments am Funktionsende.

```
void f4() {
    int j = 14;
    int* pj = new int(88);
}
```

d.) Gegeben seien die beiden folgenden Funktionen:

```
void help(int i, int& j, int* k) {
    int start = i; /* 1 */
    i = 22;
    j = 33;
    *k = 44; /* 2 */
}
void f2() {
    int a(4);
    int b = 5;
    int c(6);
    // help(&*a, &*b, &*c); // correct one
    cout << a << ", " << b << ", " << c << endl;
}
```

d1) Welcher/Welche der folgenden fünf Aufrufe der Funktion `help` ist/sind syntaktisch korrekt?

```
help(a, b, &c);
help(&a, &b, &c);
help(&a, *b, &c);
help(a, *b, c);
help(&a, *b, c);
```

d2) Veranschaulichen Sie grafisch die Stack-Speicherbelegung zum Zeitpunkt `/* 1 */`.

d3) Veranschaulichen Sie grafisch die Stack-Speicherbelegung zum Zeitpunkt `/* 2 */`.

d4) Zu welcher Bildschirmausgabe führt der Aufruf von `f2`, wenn ein richtiger Aufruf von `help` verwendet

wird? **Lösung:**

```
4,33,44
```

**Aufgabe 2 : Klassen und Erzeugung**

ca. 18 (1+1+5+1+1+2+2+4+1) Punkte

Gegeben seien die folgenden Klassendeklarationen:

```
class Furniture { // Möbel
public:
    virtual string getType() const
        {return "Unknown,";}
};

class Chair: public Furniture { // Stuhl
public:
    Chair() {datei << "+C ";}
    ~Chair() {datei << "-C ";}
    virtual string getType() const
        {return "Chair,";}
};
```

a.) Zu welcher Ausgabe führt der Aufruf von Funktion `f1`?

```
void f1(){
    datei << "Start ";
    Chair c1;
    datei << "End ";
}
```

b.) Zu welcher Ausgabe führt der Aufruf von Funktion `f2`?

```
void f2(){
    datei << "Start ";
    Chair c1;
    Chair c2(c1);
    datei << "End ";
}
```

c.) Zu welcher Ausgabe führt der Aufruf von Funktion `f3`?

```
void f3(){
    datei << "Start ";
    Furniture* f1 = new Chair();
    datei << "End ";
}
```

c2.) Zu welcher Ausgabe führt der Aufruf von Funktion `f4`?

```
void f4(){
    datei << "Start ";
    Chair* c1 = new Chair();
    datei << "End ";
}
```

c3.) Zu welcher Ausgabe führt der Aufruf von Funktion `f5`?

```
void f5(){
    datei << "Start ";
    Chair* c1 = new Chair();
    delete c1;
    datei << "End ";
}
```

c4.) Zu welcher Ausgabe führt der Aufruf von Funktion f6?

```
void f6(){
    datei << "Start ";
    Furniture* f1 = new Chair();
    delete f1;
    datei << "End ";
}
```

Eine weitere abgeleitete Klasse ist die Klasse Table.

```
class Table: public Furniture { // Tisch
public:
    Table(int le) {datei << "+T "; legs = le;}
    ~Table() {datei << "-T "};
    virtual string getType() const
        {return "Table,"};
    void setColour( string c );
private:
    int legs;
    string colour;
};
```

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

```
Table tables [12];
```

**Lösung:**

Es fehlt der Default-Konstruktor bei der Klasse. Außerdem ist bereits der Konstruktor `Vogel(int)` vorhanden, sodass kein Default-Konstruktor automatisch erzeugt wird.

e.) Implementieren Sie die Methode `setColour` in der Quellcode-Datei der Klasse. **Lösung:**

```
void Table::setColour( string c) {
    colour = c;
}
```

f.) Implementieren Sie den Kopierkonstruktor der abgeleiteten Klasse in der Quellcode-Datei der Klasse. Vermeiden Sie dabei überflüssigen Code. **Lösung:**

```
// Kopier-Konstruktor (Copy-Constructor)
Table::Table(const Table& t2): Furniture(){
    legs=t2.legs;
    colour=t2.colour;
}

// Zuweisungsoperator (Assignment-Operator)
Table& Table::operator=(const Table& t2){
    legs=t2.legs;
    colour=t2.colour;
    return *this;
}
```

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

```
#ifndef Klasse
#define Klasse
. . .
#endif
```

Außerdem gibt es eine Containerklasse zur Verwaltung der Klasse `Furniture` und der davon abgeleiteten Klassen:

```
class Room { // Raum
public:
    Room(): t1(4) {datei << "+R "};
    ~Room() {datei << "-R "};
private:
    Table t1;
    Chair chairs [4];
};
```

h.) Zu welcher Ausgabe führt der Aufruf von Funktion f7?

```
void f7(){
    datei << "Start ";
    Room r1;
    datei << "End ";
}
```

i.) Erklären Sie, warum die Initialisierungsliste im Konstruktor von `Room` erforderlich ist (Einen zusätzlichen Punkt erhalten Sie, wenn Sie die Zeichenfolge benennen, die hier als Initialisierungsliste bezeichnet wird)? **Lösung:**

Die Initialisierungsliste ist die Zeichenfolge `t1(4)` nach dem Doppelpunkt. Sie ist erforderlich, weil die Klasse `Table` keinen Default-Konstruktor besitzt. Deshalb muss der Konstruktor von `Table` explizit im Konstruktor von `Room` aufgerufen werden und zwar noch bevor der *eigentliche* Code des Konstruktors ausgeführt wird.

**Lösung:**

Ausgabe in die Datei `datei`:

```
f1
Start +C End -C
f2
Start +C End -C -C
f3
Start +C End
f4
Start +C End
f5
Start +C -C End
f6
Start +C End
f7
Start +T +C +C +C +C +R End -R -C -C -C -C -T
+T +C +C
```

### Aufgabe 3 : STL und Testen

ca. 20 (2+2+1+3+3+5+4) Punkte

Im Folgenden werden weiterhin die Klassendeklarationen der Klassen aus der vorherigen Aufgabe benötigt. Allerdings verwenden wir eine alternative Implementierung für den Container:

```
class Room1 { // Raum
public:
    Room1() {}
    ~Room1() {}
    void addFurniture(Furniture& f);
    const Furniture* getFurniture(int i) const;
    void print(ostream& datei) const;
private:
    vector<Furniture*> cont;
};
```

a.) Welche Vorteile ergeben sich aus dieser alternativen Implementierung (wenn man vom evtl. zusätzlichem Aufwand einmal absieht)? **Lösung:**  
Es können beliebig viele Möbelstücke dem Raum hinzugefügt werden, nicht nur ein Tisch und vier Stühle. Außerdem könnten auch Instanzen von weiteren Unterklassen von Furniture in den Raum gestellt werden.

b.) Warum muss als Container der Typ `vector<Furniture*>` anstatt `vector<Furniture>` verwendet werden? **Lösung:**  
Man könnte dann nicht mehr erkennen, was für ein Möbeltyp sich im Container befindet. Alle Möbelstücke, die eingefügt werden, werden in den Basistyp Furniture gewandelt.

c.) Warum ist für den Container der Typ `vector<Table*>` nicht sinnvoll? **Lösung:**  
Es könnten dann nur Tische in den Container eingetragen werden; oder man müsste fehleranfällige cast-Operationen verwenden.

d.) Implementieren Sie die restlichen Zeilen der Methode `Room1::print`.

```
// Ausgabe aller Möbel im Container durch
// Aufruf von getType in einer Schleife
// Output of all furniture in the container
// by calling getType in a loop
void Room1::print(ostream& d) const{
    vector<Furniture*>::const_iterator iter;
```

**Lösung:**

```
// Ausgabe aller Möbel im Container durch
// Aufruf von getType in einer Schleife
// Output of all furniture in the container
// by calling getType in a loop
void Room1::print(ostream& d) const{
    vector<Furniture*>::const_iterator iter;
    iter = cont.begin();
    while (iter != cont.end()) {
        d << ((*iter)->getType());
        ++iter;
    }
}
```

e.) Implementieren Sie die Methode `Room1::addFurnitureAusgabe` in die Datei `datei`:

(vermutlich nur eine Zeile Code). **Lösung:**

```
void Room1::addFurniture(Furniture& f){
    cont.push_back(&f);
}
```

f.) Warum darf der Parametertyp nicht `const Furniture&` sein? **Lösung:**

Ein Zeiger auf den Parameter wird direkt in den Container eingetragen. Über den Zeiger könnten Änderungen vorgenommen werden. Der Compiler weiß nicht, dass dies nicht erfolgt. Wenn man den Parameter also gerne als `const Furniture&` hätte, was sinnvoll wäre, müsste man den Containertyp ebenfalls ändern in: `vector<const Furniture*>`.

g.) Beschreiben Sie (nur mit Worten) einen Test für die Methode `Room1::addFurniture`.

Verwenden Sie dazu die Methode `Room1::getFurniture`:

```
// Das i-te Möbelstück im Container wird geliefert .
// return of furniture number i or NULL-Pointer
const Furniture* Room1::getFurniture(int i) const {
    // Type-Cast/Umwandlung to unsigned int
    vector<Furniture*>::size_type no = i;
    if ((no >= 0) && (no < cont.size())) {
        return cont[i];
    }
    else {
        return NULL;
    }
}
```

**Lösung:**

```
/** Ein Raum und zwei Möbelstücke
werden erzeugt. Die Möbelstücke werden
dem Raum hinzugefügt. Der Test ist
erfolgreich, wenn getFurniture die Möbel
wieder zurückliefert .
*/
```

h.) Implementieren Sie nun den soeben mit Worten beschriebenen Test in C++. **Lösung:**

```
bool addTest(){
    Room1 r;
    Table t1(4);
    r.addFurniture(t1);
    Chair c1;
    r.addFurniture(c1);

    return
        (r.getFurniture(0)->getType() == "Table,") &&
        (r.getFurniture(1)->getType() == "Chair,");
}
```

**Lösung:**

```
Output of r:
Table, Chair, Chair,
-C -C -T
+T +C -C -T
Alle Test erfolgreich

f10
+C Unknown, Chair, -C
f11
+C Unknown, Chair,
f12
+C +T Unknown, Chair, Table, Table, -T
```

**Aufgabe 4 : Polymorphie**

ca. 11 (1,5+1,5+6+2) Punkte

Im Folgenden werden weiterhin die Klassendeklarationen der Klassen aus der vorherigen Aufgabe benötigt (insbesondere die virtuellen Methoden `getType()`). Zur besseren Übersicht hier nochmals die relevanten Klassendeklarationen:

```
class Furniture { // Möbel
public:
    virtual string getType() const
    {return "Unknown,";}
};

class Chair: public Furniture { // Stuhl
public:
    Chair() {datei << "+C ";}
    ~Chair() {datei << "-C ";}
    virtual string getType() const
    {return "Chair,";}
};

class Table: public Furniture { // Tisch
public:
    Table(int le) {datei << "+T "; legs = le;}
    ~Table() {datei << "-T ";}
    virtual string getType() const
    {return "Table,";}
    void setColour( string c);
private:
    int legs;
    string colour;
};
```

a.) Zu welcher Ausgabe führt der Aufruf von `f10`? Ausgaben von Konstruktoren und Destruktoren ignorieren.

```
void f10() {
    Furniture f1;
    Chair c1;
    datei << f1.getType();
    datei << c1.getType();
}
```

b.) Zu welcher Ausgabe führt der Aufruf von `f11`? Ausgaben von Konstruktoren und Destruktoren ignorieren.

```
void f11() {
    Furniture* f1 = new Furniture();
    Chair* c1 = new Chair();
    datei << f1->getType();
    datei << c1->getType();
}
```

c.) Veranschaulichen Sie grafisch die Stack- und Heap-Speicherbelegung des folgenden Programmfragments vor der `for`-Schleife

```
void f12() {
    Furniture* pf = new Furniture();
    Chair* pc = new Chair();
    Table t(5);
    Table* pt = &t;

    Furniture* arr[]={pf, pc, &t, pt};
    // *1* Stack and Heap-Memory-Contents ???
    for (int i=0; i<4; ++i) {
        datei << arr[i]->getType();
    }
}
```

d.) Zu welcher Ausgabe führt der Aufruf von `f12`? Ausgaben von Konstruktoren und Destruktoren ignorieren.

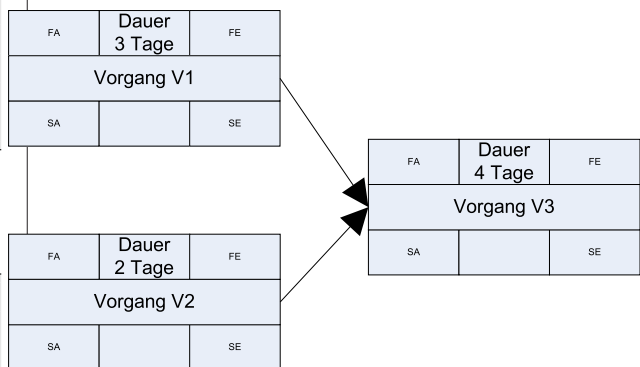
**Lösung:**

```
f10
+C Unknown, Chair, -C
f11
+C Unknown, Chair,
f12
+C +T Unknown, Chair, Table, Table, -T
```

**Aufgabe 5 : Textfragen, C++**

ca. 19 (2+1+1+6+2+2+2+1+2+2) Punkte

FA	Dauer X Tage	FE
Vorgang Vx		
SA		SE



a.) Gegeben sei obiger Netzplan mit den drei Vorgängen V1, V2 und V3. Tragen Sie in den Netzplan frühesten Anfang (FA), frühestes Ende (FE), spätesten Anfang (SA) und spätestes Ende (SE) für jeden Vorgang ein.

b.) Welche Vorgänge liegen auf dem kritischen Pfad?

c.) Geben Sie einen C++-Code-Ausschnitt an, in dem die Verwendung des Post-Increment-Operators zu einem anderen Verhalten führt als die Verwendung des Prä-Increment-Operators (am einfachsten durch Verwendung von `cout` zu zeigen).

d.) Implementieren Sie **nur** die Schnittstelle einer C++-Funktion, die sowohl das Quadrat als auch die dritte Potenz eines Eingabeparameters zurückliefert.

d2.) Handelt es sich bei dieser Funktion um eine echte Funktion oder um eine Anweisungsfunktion?

d3.) Implementieren Sie einen Test für die Funktionalität dieser Funktion.

d4.) Implementieren Sie nun die C++-Funktion (den Algorithmus) selbst.

e.) Geben Sie ein **ganz, ganz kurzes** C++-Code-Fragment an, durch dessen Ausführung sowohl Speicher auf dem Programm-Heap als auch auf dem Programm-Stack belegt wird.

f.) Geben Sie ein C++-Codefragment an, in dem zunächst eine Instanz einer Klasse X auf dem Programm-Stack erzeugt wird und anschließend auch eine Instanz von X auf dem Heap angelegt wird.

g.) Wäre Entsprechendes auch mit einem Java-Codefragment möglich? Erklären Sie!

h.) Was halten Sie von folgender Behauptung?

*Private C++-Methoden sind sinnlos, da sie nie aufgerufen werden können.*

i.) Geben Sie ein Beispiel für die Deklaration (Definition nicht erforderlich) einer Klasse mit einer minimalen Standard-Schnittstelle an. **Lösung:**

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

### Aufgabe 6 : Textfragen, Team

ca. 16 (1+2+1+2+1,5+1+1+2+2+2,5) Punkte

a.) Welcher Schritt folgt im Wasserfallmodell direkt auf den Entwurf?

b.) Warum sinkt im Allgemeinen der sog. *Truck Factor*, wenn die XP-Basistechnik *der gemeinsamen Verantwortung* angewendet wird?

c.) Ihr Programm enthält einige Programmfragmente mehrfach. Was schlagen Sie vor, um diesen Missstand zu beheben (ein *richtiges* Wort reicht)? **Lösung:** Refactoring

d.) Warum ist redundanter Code (mehrfach vorhandener identischer Code) von Nachteil für den weiteren Projektfortschritt? **Lösung:**

Er muss mehrfach getestet werden, eine Änderung ist an mehreren Stellen durchzuführen. Er macht das Programm auch unübersichtlicher. Man braucht zwei Blicke und nicht einen, um das Programm zu verstehen.

e.) Bei der Planung von Software-Projekten spielen vier Variablen eine Rolle. Eine Prozessvariable sind die *Kosten*. Wie heißen die anderen drei? **Lösung:** Umfang, Kosten, Zeit und Qualität.

f.) Was bedeutet die Technik des *Wetters von gestern* im Zusammenhang (Kontext) einer Iterationsplanung?

**Lösung:**

Nehmen Sie als Grundlage für Ihre Planung an, dass Sie diese Woche genauso viel erledigen werden wie letzte Woche.

g.) Geben Sie *Brooks-Gesetz* an. **Lösung:**

Adding men to a late project makes it later.

Einem ohnehin schon verspäteten Projekt Personal hinzuzufügen, verspätet es noch mehr.

h.) Erklären Sie es (es hört sich zunächst merkwürdig an). **Lösung:**

Die Einarbeitung des zusätzlichen Personals und der zusätzliche Kommunikationsaufwand wird zusätzliche Personalressourcen verbrauchen.

i.) Welche Vorteile ergeben sich, wenn man die Tests noch vor Beginn der Implementierung der eigentlichen Funktionalität spezifiziert (wenige kurze Sätze)?

**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.

j.) Das Team hatte sich in der letzten Iteration (Dauer 25 Arbeitstage) Aufgaben im Umfang von 12 idealen Tagen vorgenommen. Erledigt wurden allerdings nur Aufgaben im Umfang von 9 idealen Tagen. Zusätzlich wurden ungeplante Aufgaben im Umfang von 4 idealen Tagen erledigt. Berechnen Sie den Load Factor (Rechenweg angeben).

Die neue Iteration dauert 10 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 :  $\frac{9}{25} = 36\%$ , damit kann man sich in der neuen Iteration Aufgaben im Umfang von ca.  $(10 * \frac{9}{36} =)$  3,6 Tagen vornehmen.