Name:

Prof. Dr.-Ing. Hartmut Helmke
Ostfalia
Hochschule für angewandte Wissenschaften
Fakultät für Informatik

Matrikelnummer:	*Punktzahl:*
Ergebnis:	

Freiversuch ☐ *F1* ☐ *F2* ☐ *F3* ☐

Klausur im SS 2010:

Programmierkonzepte

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 !	

Bitte notieren Sie auf **allen** 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 Programmfragmenten 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.

In vielen Fällen können Sie die Lösung direkt auf dem Aufgabenblatt notieren. Falls der Platz nicht ausreichen sollte, verweisen Sie **per Pfeil** auf die Extrablätter.
Gehen Sie davon aus, dass `double` 8 Bytes sowie `int` und Zeiger jeweils 4 Bytes im Speicher belegen.

Geplante Punktevergabe

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

Punktziel	Sonderpunkte	erreicht
A1: 14: (1+3+2+2+ 2+1+3) P.		
A2: 9: (5*1 + 4 Gesamtverständnis) P.		
A3: 16: (6*2+ 3 + 1) P.		
A4: 17: (3*3+ 6 + 2) P.		
A5: 30: (2*2 + 7*1,5 +1+2*1,5 +1+2 +2+2,5+4) P.		
A6: 14: (2 * (2+2+3)) P.		
Sonderpunkte Übungen		xxxx
Summe 100 Punkte		

Aufgabe 1 : Textfragen, Team

ca. 14: (1+3+2+2+ 2+1+3) Punkte

a.) Wofür steht die Abkürzung „XP“ im Zusammenhang mit Software-Prozessmodellen?

(*——- 2 Worte hier ———*)

b.) Sie und Ihr Team gehen nach XP vor. Ihre Aufgabe ist, heute eine doppelverkettete Liste auf Basis einer bereits vorhandenen einfach verketteten Liste zu implementieren. Hierzu spezifizieren Sie die Methoden. Anschließend beschreiben Sie diese in natürlicher Sprache und implementieren Tests für die verschiedenen Methoden der Klasse. Welche Schritte folgen nun noch **in diesem konkreten Beispiel**, bevor Sie den Code dem Rest des Teams durch Einchecken zur Verfügung stellen können?

(*——- Mehrere Schritte, wobei Stichworte hier genügen ———*)

c.) Welche Basistechnik von Extreme Programming ist ohne ein Versionsverwaltungssystem wie z.B. SVN nur mit ganz hohem Aufwand umsetzbar?
(*——- Kurze Antwort mit Begründung hier ———*)

d.) Was bedeutet Refactoring?
(*——- Kurze Antwort hier ———*)

e.) Sie wollen die XP-Basistechnik „Zwei-Leute-ein-Bildschirm“ einsetzen und Ihren Chef davon überzeugen. Warum könnte Ihr Chef dagegen sein? Was antworten Sie ihm, um ihn von den Vorteilen von „Zwei-Leute-ein-Bildschirm“ zu überzeugen?
(*——- Kurze Antwort hier ———*)

f.) Sie vervierfachen die Zeit eines Projekts, d.h. z.B. eine Dauer von acht statt zwei Monaten. Welche Auswirkungen auf die Variable Umfang könnte dieses haben?
(*——- Kurze Antwort hier ———*)

g.) Von wem stammt das *Gesetz* „Adding men to a late project makes it later, not earlier“[1]
(*——- Kurze Antwort hier ———*)

Erklären Sie, warum das so ist. Sollte man deshalb grundsätzlich ein Projekt besser mit drei Leuten als mit 200 durchführen?

(*——- Kurze Antwort hier ———*)

[1] Einem bereits verspätetem Projekt Leute hinzuzufügen, verspätet es noch mehr.

Aufgabe 2 : Polymorphie

ca. 9: (5*1 + 4 Gesamtverständnis) Punkte

Gegeben sei die folgende Deklaration der Klasse `Hose`

```
class Hose {
public:
   Hose(int k=0) {knoepfe = k;}
   ~Hose()       {;}

    virtual  int knopfAnzahl() const { return knoepfe;}
    string  farbe() {return "gruen";}

private:
   int knoepfe;
};
```

und die abgeleitete Klasse `Jeans`:

```
class Jeans: public Hose {
public:
   Jeans(): Hose(400) {}

    virtual  int knopfAnzahl() const { return 4;}
    string  farbe() {return "blau";}
};
```

a.) Welche Ausgabe in `datei` liefert der Aufruf von `abgeleitet1`?

```
void abgeleitet1 (){
   Hose* j1 = new Jeans;
   datei << j1->knopfAnzahl();
   datei << " " << j1->farbe();
}
```

(*——- Lösung hier notieren ———*)

b.) Welche Ausgabe in `datei` liefert der Aufruf von `abgeleitet2`?

```
void help(Hose* h2) {
   datei << h2->knopfAnzahl();
   datei << " " << h2->farbe();
}
void abgeleitet2 (){
   Hose* h1 = new Jeans;
   help(h1);
}
```

(*——- Lösung hier notieren ———*)

c.) Welche Ausgabe in `datei` liefert der Aufruf von `abgeleitet3`?

```
void helpWert(Hose wert) {
   datei << wert.knopfAnzahl();
   datei << " " << wert.farbe();
}
void abgeleitet3 (){
   Hose* h1 = new Jeans();
   helpWert(*h1);
}
```

(*——- Lösung hier notieren ———*)

d.) Welche Ausgabe in `datei` liefert der Aufruf von `abgeleitet4`?

```
void abgeleitet4 (){
   Jeans* jp1 = new Jeans();
   datei << jp1->knopfAnzahl();
   datei << " " << jp1->farbe();
   delete jp1;
}
```

(*——- Lösung hier notieren ———*)

e.) Welche Ausgabe in `datei` liefert der Aufruf von `abgeleitet5`?

```
void abgeleitet5 (){
   Jeans j2;
   datei << j2.knopfAnzahl();
   datei << " " << j2.farbe();
}
```

(*——- Lösung hier notieren ———*)

Aufgabe 3 : Werte- und Referenzsemantik

ca. 16: (6*2+ 3 + 1) Punkte

Gegeben sei die folgende Deklaration der Klasse `Hose`.

```
class Hose {
public:
   Hose(int k=0) {knoepfe = k;}
   ~Hose()       {;}

    virtual  int knopfAnzahl() const { return knoepfe;}
    string  farbe() {return "gruen";}

private:
   int knoepfe;
};
```

a.) Welche Ausgabe in `datei` liefert der Aufruf von `callHose1`?

```
void FunkHose1(Hose h1) {
   Hose hx(44);
   h1 = hx; /* 1 */
}
void callHose1() {
   Hose hc(21);
   datei << hc.knopfAnzahl() << " ";
   FunkHose1(hc);
   datei << hc.knopfAnzahl() << " ";
}
```

(*——- Lösung hier notieren ———*)

b.) Veranschaulichen Sie im obigen Programmfragment die Speicherbelegung im Stackspeicher unmittelbar nachdem die mit /* 1 */ gekennzeichnete Anweisung ausgeführt wurde.
(Speicherbelegung):

5000:	
5004:	
5008:	
5012:	
5016:	

Stack-Speicher

c.) Welche Ausgabe in `datei` liefert der Aufruf von `callHose2`?

```
void FunkHose2(Hose& h1) {
   Hose hx(44);
   h1 = hx; /* 1 */
}
void callHose2() {
   Hose hc(21);
   datei << hc.knopfAnzahl() << " ";
   FunkHose2(hc);
   datei << hc.knopfAnzahl() << " ";
}
```

(*——- Lösung hier notieren ———*)

d.) Veranschaulichen Sie die Speicherbelegung im Stackspeicher unmittelbar nachdem die mit /* 1 */ gekennzeichnete Anweisung ausgeführt wurde.
(Speicherbelegung):

5000:	
5004:	
5008:	
5012:	
5016:	

Stack-Speicher

e.) Welche Ausgabe in `datei` liefert der Aufruf von `callHose3`?

```
void FunkHose3(Hose* h1) {
   Hose hx(44);
   *h1 = hx; /* 1 */
}
void callHose3() {
   Hose hc(21);
   datei << hc.knopfAnzahl() << " ";
   FunkHose3(&hc);
   datei << hc.knopfAnzahl() << " ";
}
```

(*——- Lösung hier notieren ———*)

f.) Veranschaulichen Sie die Speicherbelegung im Stackspeicher unmittelbar nachdem die mit /* 1 */ gekennzeichnete Anweisung ausgeführt wurde.
(Speicherbelegung):

5000:	
5004:	
5008:	
5012:	
5016:	

Stack-Speicher

g.) Beschreiben Sie kurz (2 Sätze reichen vermutlich) die Vorteile der Strategie `Test First`.

(*——- Lösung hier notieren ———*)

h.) Hätte die Methode `farbe` auch als konstante Methode vereinbart werden dürfen? Warum?

(*——- Lösung hier notieren ———*)

Aufgabe 4 : Testen

ca. 17: (3*3+ 6 + 2) Punkte

a.) Im Folgenden ist die Spezifikation eines Tests für die Funktion `erzeugeHose` angegeben (Klasse `Hose` wie vorherige Aufgabe).

```
/* Die Funktion prüft, ob die Funktion erzeugeHose
 korrekt arbeitet, d.h. ob sie zwei neue Instanzen von
Hose mit Werten wert1 und wert2 auf dem Heap erzeugt,
sodass z1 bzw. z2 anschließend darauf verweisen.
Der Test ruft erzeugeHose mit verschiedenen
Werten auf. Nur wenn alle Prüfungen erfolgreich sind,
wird true geliefert. */
```

Die Implementierung des Tests liegt auch bereits vor.

```
bool testErzeugeHose(){
   bool retValue = true;
   Hose* z1=NULL; Hose* z2=NULL;
   int w1 = 22; int w2 = 123;
   // Speicherbelegung in der Funktion (siehe später)
   erzeugeHose(z1, z2, w1, w2);
   retValue = retValue &&
      z1->knopfAnzahl()==w1 &&
      z2->knopfAnzahl()==w2;
   w1 = -22; w2 = -123;
   erzeugeHose(z1, z2, w1, w2);
   retValue = retValue &&
      z1->knopfAnzahl()==w1 &&
      z2->knopfAnzahl()==w2;
   w1 = 400; w2 = w1;
   erzeugeHose(z1, z2, w1, w2);
   retValue = retValue &&
      z1->knopfAnzahl()==w1 &&
      z2->knopfAnzahl()==w2;
   return retValue;
}
```

Ihre Aufgabe ist nun zunächst die Schnittstelle (Prototyp oder auch Funktionsdeklaration genannt) dieser Funktion `erzeugeHose` anzugeben.

(*——- Lösung hier notieren ———*)

b.) Bevor Sie mit der Implementierung der Funktionalität beginnen, erklären Sie **an diesem Beispiel,** was die Technik `Think, Red-Bar, Green-Bar, Refactor` bedeutet.

(*——- Lösung hier oder auf Extrablatt notieren ———*)

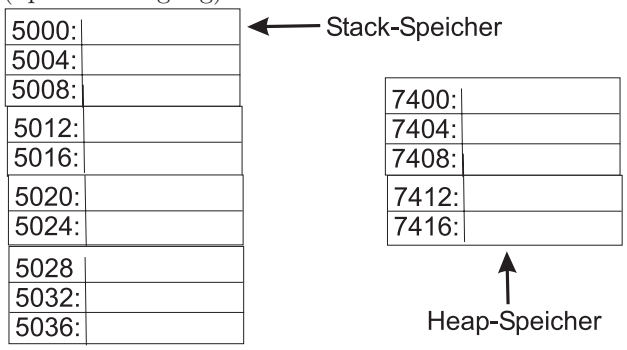
c.) Implementieren Sie die Funktion `erzeugeHose`, sodass der Test `testErzeugeHose` erfolgreich ausgeführt wird [2].

(*——- Lösung hier oder auf Extrablatt notieren ———*)

d.) Veranschaulichen Sie die Speicherbelegung (Stack- und Heapspeicher) beim ersten Aufruf von `erzeugeHose` am Ende der Funktion `erzeugeHose`. Hier noch einmal das relevante Codesegment:

```
Hose* z1=NULL; Hose* z2=NULL;
int w1 = 22; int w2 = 123;
// Speicherbelegung in der Funktion (siehe später)
erzeugeHose(z1, z2, w1, w2);
```

(Speicherbelegung):



e.) Begründen Sie, warum es in der vorliegenden Implementierung des Tests `testErzeugeHose` Speicherlecks gibt. Wie viele Instanzen von `Hose` werden insgesamt im Test nicht freigegeben?
(*——- Lösung hier notieren ———*)

[2] Wenn Sie in der vorherigen Teilaufgabe bereits die Schnittstelle angegeben haben, reicht hier der Funktionsrumpf, d.h. alles zwischen der öffnenden und schließenden geschweiften Klammer.

Aufgabe 5 : Konstruktor/Destruktor und Stack-/ Heapspeicherbelegung

ca. 30: (2*2 + 7*1,5 +1+2*1,5 +1+2 +2+2,5+4) Punkte
Gegeben sei die folgende Deklaration der Klasse `Hose`.

```
class Hose {
public:
   Hose(int k=0){knoepfe = k; datei<<"+H "<<knoepfe;}
   ~Hose()      {datei << " -H " << knoepfe;}

    virtual int knopfAnzahl() const { return knoepfe;}
    string farbe() {return "gruen";}

private:
   int knoepfe;
};
```

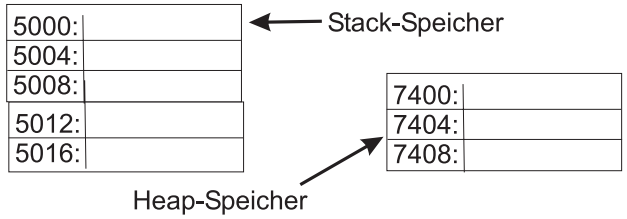
Es geht im Folgenden jeweils darum, welche Instanzen der Klasse `Hose` wann erzeugt und gelöscht werden, d.h. wann werden welche Konstruktoren und Destruktoren aufgerufen. Vergessen Sie nicht den String `Finito` bei der Ausgabe!
Leerzeichen und Leerzeilen werden bei der Bewertung nicht beachtet.

a.) Welche Ausgabe in `datei` liefert der `funk1`-Aufruf?

```
void funk1() {
   Hose* ph1 = new Hose(4);
   Hose* ph2 = new Hose(2);
   datei << " Finito ";
}
```

(*——- Lösung hier notieren ———*)

b.) Veranschaulichen Sie die Speicherbelegung der obigen Funktion im Stack- und Heapspeicher unmittelbar nach Ausgabe des Strings `Finito` . Eine Instanz der Klasse `Hose` belegt 4 Byte im Speicher. Zeiger belegen ebenfalls 4 Byte.
(Speicherbelegung):

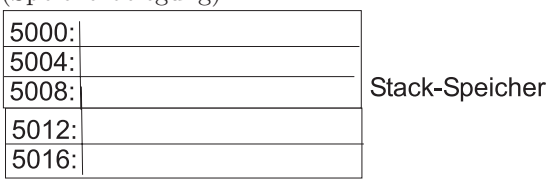


c.) Welche Ausgabe in `datei` liefert der `funk2`-Aufruf?

```
void funk2() {
   Hose h1(4);
   Hose h4(2);
   datei << " Finito ";
}
```

(*——- Lösung hier notieren ———*)

d.) Veranschaulichen Sie die Speicherbelegung der obigen Funktion im Stackspeicher unmittelbar nach Ausgabe des Strings `Finito` .
(Speicherbelegung):

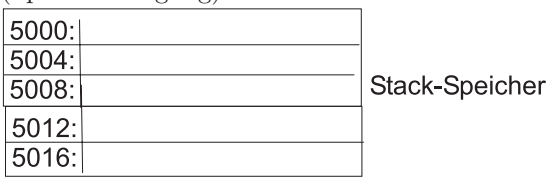


e.) Welche Ausgabe in `datei` liefert der `funk3`-Aufruf?

```
void funk3() {
   Hose* ph1=NULL;
   Hose* ph2=ph1;
   datei << " Finito ";
}
```

(*——- Lösung hier notieren ———*)

f.) Veranschaulichen Sie die Speicherbelegung der obigen Funktion im Stackspeicher unmittelbar nach Ausgabe des Strings `Finito` .
(Speicherbelegung):



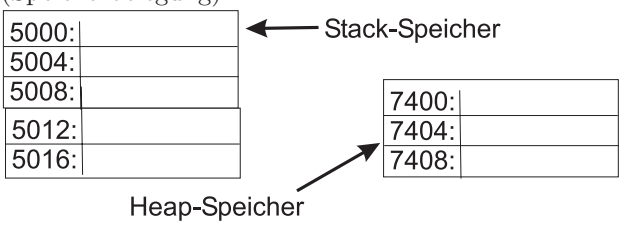
g.) Welche Ausgabe in `datei` liefert der `funk4`-Aufruf?

```
void funk4() {
   Hose* ph = new Hose(2);

   // entspricht  Hose h2(*ph);
   Hose h2 = *ph; //
   delete ph;
   datei << " Finito ";
}
```

(*——- Lösung hier notieren ———*)

h.) Veranschaulichen Sie die Speicherbelegung der obigen Funktion im Stack- und Heapspeicher unmittelbar **vor** dem `delete`-Aufruf.
(Speicherbelegung):



i.) Welche Ausgabe in `datei` liefert der `funk5`-Aufruf?

```
void funk5() {
   Hose hosen[3];
   datei << " Finito ";
}
```

(*——- Lösung hier notieren ———*)

j.) Im Gegensatz zur Klasse `Hose` können von der

folgenden Klasse `Tier` keine Arrays erzeugt werden (Syntaxfehler).Warum?

```
class Tier {
public:
   Tier(int f) {fuesse = f;}
   ~Tier()     {fuesse=2;}
private:
   int fuesse;
};
```

(*——- Lösung hier notieren ———*)

k.) Die folgende Funktion ist aber zumindest syntaktisch richtig. Warum? Können etwa doch Instanzen der Klasse `Tier` erzeugt werden?

```
void funk5a() {
   Tier* tiere[3];
}
```

(*——- Lösung hier notieren ———*)

l.) Welche Ausgabe in `datei` liefert der `funk6`-Aufruf?

```
void funk6() {
   Hose* hosen[3];
   datei << " Finito ";
}
```

(*——- Lösung hier notieren ———*)

m.) Welche Ausgabe in `datei` liefert der `funk7`-Aufruf?

```
void funk7() {
   Hose* hosen[3];
   hosen[1] = new Hose(4);
   datei << " Finito ";
}
```

(*——- Lösung hier notieren ———*)

n.)
Zur Unterstützung der Heap-Freigabe ist die Klasse `AutoPtr` wie angegeben definiert.

```
class AutoPtr{
public:
   AutoPtr(Hose* p) {ph = p;}
   ~AutoPtr() {delete ph;}
private:
   Hose* ph;
   };
```

Welche Ausgabe in `datei` liefert der `funk8`-Aufruf?

```
void funk8() {
   AutoPtr aph1(new Hose(11));
   AutoPtr aph2(new Hose(12));
   datei << " Finito ";
}
```

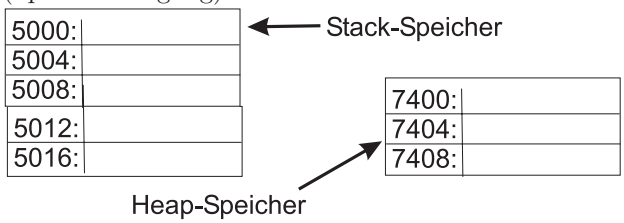
(*——- Lösung hier notieren ———*)

o.) Welche Ausgabe in `datei` könnte der `funk9`-Aufruf liefern?

```
void funk9() {
   AutoPtr aph1(new Hose(11));
   AutoPtr aph2 = aph1;
   datei << " Finito ";
}
```

(*——- Lösung hier notieren ———*)

p.) Veranschaulichen Sie die Speicherbelegung der obigen Funktion im Stack- und Heapspeicher unmittelbar nach Ausgabe des Strings `Finito` .
(Speicherbelegung):



q.) Welchen Fehler enthält die Klasse `AutoPtr`, der zu einem undefinierten Verhalten des Funktionsaufrufs von `funk9` führt?
Korrigieren Sie diesen Fehler, d.h. erweitern/-verändern Sie die Klasse `AutoPtr`, sodass zumindest `funk9` ein definiertes Verhalten zeigt.
(*——- Lösung hier oder auf Extrablatt notieren ———*)

Aufgabe 6 : Kopieren und Zuweisen

ca. 14: (2 * (2+2+3)) Punkte
Gegeben sei die folgende Deklaration der Klasse `Hose`.

```
class Hose {
public:
   Hose(int k=0) {knoepfe = k;}
   ~Hose()       {;}

    virtual  int knopfAnzahl() const { return knoepfe;}
    string  farbe() {return "gruen";}

private:
    int knoepfe;
};
```

a.) Implementieren Sie für diese Klasse den Zuweisungsoperator.

(*——- Lösung hier notieren ———*)

b.) Spezifizieren Sie (mit deutschen Worten) einen Test, der die Funktionalität Ihrer Implementierung zeigt/nachweist.

(*——- Lösung hier notieren ———*)

c.) Implementieren Sie nun diesen soeben spezifizierten Test.

(*——- Lösung hier notieren ———*)

d.) Implementieren Sie für diese Klasse den Kopier-Konstruktor.

(*——- Lösung hier notieren ———*)

e.) Spezifizieren Sie (mit deutschen Worten) einen Test, der die Funktionalität Ihrer Implementierung zeigt/nachweist.

(*——- Lösung hier notieren ———*)

f.) Implementieren Sie nun diesen soeben spezifizierten Test.

(*——- Lösung hier notieren ———*)