Prof. Dr. S. Naumann | M. Schmidt, M.Sc

Programmierung III (PROGRA III)

Klausur Sommersemester 2011
23. März 2011

Bachelor-Studiengang
O Angewandte Informatik O Medieninformatik O _____

Name _____

Matrikel-Nr. _____

Wichtige Hinweise:

- Sie haben *90 Minuten* Zeit für die Bearbeitung der Klausur
- Prüfen Sie die Klausur auf *Vollständigkeit*! Sie umfasst insgesamt 10 Seiten
- Lassen Sie die Klausur *zusammengeheftet*
- Es sind *keine Hilfsmittel* zugelassen
- Benutzen Sie einen *dokumentenechten Stift*
- Nur *lesbare*, *nachvollziehbare* und (falls gefordert) *begründete* Lösungen werden gewertet
- Kreuzen Sie unten an, welche Aufgaben Sie bearbeitet haben
- Die Blätterrückseiten können genutzt werden. Bitte verweisen Sie aber darauf.
- Weitere Blätter erhalten Sie auf Anfrage. Bitte auf zusätzlichen Blättern Aufgabennummer und Matrikelnummer angeben!
- Die Aufgaben sind unabhängig voneinander. Bitte beachten Sie, dass es unterschiedlich viele Punkte gibt!

Punkte:

Aufgabe	1	2	3	4	5	6	7	8	9	10	11	12	∑
Erreichbare Punkte	5	8	5	8	5	15	5	5	8	5	6	25	100
Erzielte Punkte													

Viel Erfolg!

Aufgabe 1 (Allgemeine Fragen) (5 Punkte)

a) Beantworten Sie die folgenden Fragen mit ja oder nein. Achtung: Falsche Antworten führen zu Punktabzug! Insgesamt sind jedoch mindestens 0 Punkte erreichbar.

	ja	nein
Typische Kennzeichen einer objektorientierten Programmiersprache sind Klassen, Vererbung und Polymorphismus		
Objektorientierte Programmierung ist besonders für das „Programmieren im Großen“ geeignet		
„Zustände“ und „Methoden“ sind Kernbestandteile eines Objekts		
UML ist eine der ersten objektorientierten Programmiersprachen		
Ein Objekt kann aus mehreren Klassen erzeugt werden		

Aufgabe 2 (Schlüsselwort const) (2+6 Punkte)

a) Wofür wird in C++ das Schlüsselwort const benötigt?

b) Erläutern Sie die folgenden Definitionen!

```
const double EULER = 2.72;
```

```
const double * const EULER_P = &EULER:
```

```
const char nothing;
```

Aufgabe 3 (Allgemeine Fragen) (5 Punkte)

Beantworten Sie die folgenden Fragen mit ja oder nein. Achtung: Falsche Antworten führen zu Punktabzug! Insgesamt sind jedoch mindestens 0 Punkte erreichbar.

	ja	nein
Ausschließlich ein überladener Konstruktor darf anders als die zugehörige Klasse heißen		
In der Deklaration `char const *buf`; ist `buf` ein Zeiger auf konstante chars.		
`class` ist nur ein anderes Schlüsselwort für `struct`. Die Funktionsweise ist identisch.		
Für einen benutzerdefinierten Datentyp `Person` schafft die Definition `Person p1;` automatisch den notwendigen Speicherplatz auf dem Heap		
Virtuelle Destruktoren dürfen mit Parametern überladen werden		

Aufgabe 4 (Klassendefinition) (8 Punkte)

Gegeben ist folgende Klassendefinition in der Header-Datei:

```
class X {
public:
   int a, *b;
   long c, *d;
   X(int d=0);
};
```

Beurteilen Sie die folgenden Methodenimplementierungen in der cpp-Datei. Begründen Sie nicht korrekte Fälle.

	korrekt	nicht korrekt	Begründung bei „nicht korrekt“
`X(int d=0) {}`			
`X::X(int e):a(e) { b=&e;};`			
`X::X() { a=0;};`			
`X::~X() { delete a;};`			
`X::~X(int e) { delete d;};`			
`X::~X() { delete [] d;};`			

Aufgabe 5 (Zuweisen und kopieren) (5 Punkte)

Beantworten Sie die folgenden Fragen mit ja oder nein. Achtung: Falsche Antworten führen zu Punktabzug! Insgesamt sind jedoch mindestens 0 Punkte erreichbar.

	ja	nein
Der Kopierkonstruktur erzeugt ein neues Objekt und kopiert in dieses neue Objekt die Werte des übergebenen Objekts		
Beim überladenen Zuweisungsoperator muss ein Rückgabewert nur dann vorgesehen werden, wenn die Klasse „Zeiger auf Zeiger"-Datenelemente enthält (**)		
Existiert ein Kopierkonstruktor in einer Klasse, so wird kein überladener Zuweisungsoperator mehr benötigt.		
Mit `static` deklarierte Variablen haben für alle Objekte der Klassen nur einen Wert		
Der Kopierkonstruktor wird implizit bei der Call-by-Reference Parameterübergabe von Objekte aufgerufen		

Aufgabe 6 (Klassendeklaration) (10+5 Punkte)

Schreiben Sie, basierend auf den vorgegebenen Attributen, eine vollständige Klassendeklaration für eine abstrakte Klasse `Buch` mit allen mindestens verfügbaren Methoden sowie einer Methode `print()`. Sie soll zudem von der aus der Vorlesung bekannten Klasse `Sortable` abgeleitet werden.

```
string titel;
char ** autoren;
float preis;
```

b) Die Klasse `Buch` ist von der Klasse `Sortable` abgeleitet. Definieren Sie die noch fehlende Methode, um zwei Objekte der Klasse `Buch` miteinander vergleichen zu können. Gehen Sie bei Ihrer Implementierung vereinfacht davon aus, dass Bücher dann gleich sind, wenn ihre ISBN gleich ist.

Aufgabe 7 (Konstruktoren) (5 Punkte)

Betrachten Sie die folgenden Code-Ausschnitte:

```
1: // Programm 1
2: String getline()
3: {
4:  char buf [100];
5:  gets(buf);
6:  String  ret = buf;
7:  return ret;
8: }
```

```
1: // Programm 2
2: String getline()
3: {
4:  char buf [100];
5:  gets(buf);
6:  return buf;
7: }
```

Benennen Sie alle Programmzeilen in den beiden Programmen, an denen ein Konstruktor von `string` aufgerufen wird. Um welchen Konstruktor handelt es sich jeweils (Begründung)?

Aufgabe 8 (Vererbung) (5 Punkte)
Beantworten Sie die folgenden Fragen mit ja oder nein. Achtung: Falsche Antworten führen zu Punktabzug! Insgesamt sind jedoch mindestens 0 Punkte erreichbar.

	ja	nein
Vererbung kennzeichnet eine „ist-ein“-Beziehung		
In Oberklassen können Datenelemente der Unterklasse wie private Datenelemente der Unterklasse verwendet werden		
Klassen, die Objekte anderer Klassen speichern und verwalten sollen, werden Behälterklassen (engl. Container Class) genannt		
Nur die Basisklasse in einer Vererbungshierarchie kann abstrakt sein		
In Unterklassen können alle Datenelemente der Oberklasse wie private Datenelemente der Unterklasse verwendet werden		

Aufgabe 9 (private-Ableitung) (8 Punkte)

Eine Klasse kann in seltenen Fällen nicht nur `public` von einer anderen abgeleitet werden, sondern auch `private`. Das bedeutet, dass alle Methoden und Attribute aus Sicht der abgeleiteten Klasse `private` sind. Innerhalb der Implementierung der abgeleiteten Klasse gelten allerdings noch die bestehenden Zugriffsregeln der Oberklasse. Beim Zugriff von außerhalb ist alles `private`.

```
class Basis
{    private: int var_1;
     protected: int var_2;
     public: int var_3; };

class Abgelitten : private Basis
{   int f1() {return var_1;} // (1)
    int f2() {return var_2;} // (2)
    int f3() {return var_3;} // (3) };

int main()
{   int i;
    Abgelitten a;
    Basis b;
    i = a.var_3;  // (4)
    i = b.var_3;  // (5)
    b = a;        // (6) };
```

Welche der Operationen an den nummerierten Stellen im Programm sind erlaubt, welche sind nicht erlaubt? (Begründung!)

Aufgabe 10 (Polymorphismus) (5 Punkte)
Beantworten Sie die folgenden Fragen mit ja oder nein. Achtung: Falsche Antworten führen zu Punktabzug! Insgesamt sind jedoch mindestens 0 Punkte erreichbar.

	ja	nein
Ist eine Methode in der Basisklasse als virtuell deklariert, muss sie in abgeleiteten Klassen auch implementiert werden		
Wenn zur Übersetzungszeit (also vom Compiler) festgelegt wird, welche Funktion einer Klasse aufzurufen ist, dann wird dies dynamisches/spätes Binden genannt		
Eine virtuelle Funktion ist eine Funktion, deren Code vom Compiler generiert wird		
Aus einer abstrakten Klasse können keine Objekte erzeugt werden		
Enthält eine Klasse ausschließlich rein virtuelle Methoden, ist sie abstrakt		

Aufgabe 11 (Templates) (3+3 Punkte)

a) Betrachten Sie den folgenden Programmausschnitt.

```
template <class T>
int MyMax(T a, int n)
{     int max = 0;
      for(int i = 0; i < n ;i++){
            (a[i] > a[max])? max=i: max=max;
      }
      return max; }
```

a) Was macht die Funktion und warum wird hier mit einem Template gearbeitet?

b) Für den Datentyp char ** soll eine Ausnahme definiert werden. Wie sehen die ersten beiden Zeilen aus (Templatedefinierung und Funktionskopf)?

Aufgabe 12 (Immobilienverwaltung) (8+10+7 Punkte)
Eine Immobilienfirma verwaltet Wohnungen und Häuser. Ein *Wohnhaus* hat eine Wohnfläche, den Kaufpreis pro qm und einen endgültigen Kaufpreis. Der Kaufpreis errechnet sich aus dem Produkt von Wohnfläche und Preis pro Quadratmeter. Der Einfachheit halber wird die Wohnfläche anhand von zwei Eckpunkten errechnet und gespeichert (vgl. Übungsaufgabe „Grundstück").
Eine *Mietwohnung* hat neben ihrer Wohnfläche noch eine Warmmiete und eine Kaltmiete. Die Kaltmiete berechnet sich aus dem Produkt von Fläche der Wohnung und Preis pro Quadratmeter. Die Warmmiete berechnet sich aus der Kaltmiete, auf die noch einmal 30% ihres Betrags aufgeschlagen werden.

a) Deklarieren Sie die Basisklasse „Immobilie".

```
#include <iostream>
using namespace std;
```

b) Implementieren Sie die Klassen Wohnhaus und Mietwohnung, jeweils einschließlich eines überladenen Konstruktors.
Wohnhaus.h:

```
#include <iostream>
using namespace std;
```

Wohnhaus.cpp:

Mietwohnung.h:

```
#include <iostream>
using namespace std;
```

Mietwohnung.cpp:

c) Schreiben Sie ein Hauptprogramm, bei dem jeweils ein Wohnhaus und eine Mitwohnung erzeugt werden (auf dem Heap) und geben Sie jeweils die Objektwerte aus.
main.cpp: