Fachhochschule
Ravensburg-Weingarten

Prof. Gampp
11. Juli 2002

Name: _____

Vorname: _____

Matr.-Nr.: _____

Ich bin einverstanden, dass meine Note unter Angabe der Matrikelnummer auf einem öffentlichen Notenaushang erscheint.

Unterschrift: _____

Klausur Objektorientierte Programmierung

(Kenn-Nummern 1675, 1805, 2144, 4011)

Aufgabe 1 (18 Punkte)

Das folgende Quellfile wird gemäß ANSI-C++-Sprachstandard compiliert und gelinkt. Welche Fehler treten dabei auf? Markieren Sie die für Fehler verantwortlichen Stellen und geben Sie neben dem Code oder auf der Rückseite eine kurze stichwortartige Fehlerbeschreibung!

Logische oder Laufzeit-Fehler sind nicht gefragt.

```
#include <iostream>
using namespace;

class C {
  protected:
    int a;
  public:
    void f();
    virtual C() {
      a = b;
    }
};
void f() {
  C c = new C;
}
class D: public C {
  public:
    int b;
    b = 0;
}
void C::f(int b) {
  a = 17;
}
int main() {
  C x;
  cout << x << this;
  return x.a;
}
```

Aufgabe 2 (18 Punkte)

Es liegen folgende (fehlerfreien) Klassendefinitionen vor:

```
class A {
  public:
    int i;  // der Einfachheit halber public
    A() { i = 1;}
    void access(int p = 5) {                    // in Teil c) virtuell
      add(p);
    }
    void add(int p) { i = i+p;}                 // in Teil c) virtuell
};
class B : public A {
    float k;
  public:
    B(int _k) { k = _k;}
    void add(int p) { i = i+p+k;}
};
```

a) Diese Definitionen werden in dem folgenden Codefragment verwendet. Tragen Sie in die Kästchen neben dem Code ein, welche Ausgaben entstehen!

```
A obj;                                  // in Teil b) B obj(2)
obj.add(3);
cout << obj.i << endl;
obj.access();
cout << obj.i << endl;
```

b) Die erste Zeile des Codefragments aus Teil a) wird ersetzt durch `B obj(2)`. Tragen Sie in der linken Spalte der nachfolgenden Tabelle ein, welche Elementfunktionen aus `A` und `B` nacheinander aufgerufen werden wenn das geänderte Codestück abläuft!
Geben Sie die Reihenfolge korrekt wieder (sie ist durch den Ausführungszeitpunkt der ersten Anweisung jeder Funktion festgelegt) und machen Sie die Klassenzugehörigkeit der Elementfunktionen durch den Gültigkeitsbereichsoperator deutlich!

Aufrufreihenfolge für Aufgabe b):	Aufrufreihenfolge für Aufgabe c):

c) Die Funktionen `access` und `add` in Klasse `A` sollen jetzt als virtuell deklariert werden. Analysieren Sie auf gleiche Weise wie in b) die von folgendem Codefragment erzeugten Aufrufe:

```
A* p = new B(2);
p->add(4);
cout << p->i << endl;
p->access(1);
cout << p->i << endl;
```

Tragen Sie die aufgerufenen Elementfunktionen in die rechte Spalte der vorstehenden Tabelle und die erzeugte Ausgabe wieder in die Kästchen neben dem Code ein!

Aufgabe 3 (12 Punkte)

a) Ergänzen Sie die Definition einer Klasse `C`, die von der Klasse `B` aus Aufgabe 2 abgeleitet ist und einen parameterlosen Konstruktor besitzt, welcher den aktuellen Wert der Elementvariablen `i` mit genau drei Nachkommastellen ausgibt:

```
class C : public B          {
  public:
    C                       {
        cout <<
    }
};
```

b) Ergänzen Sie die Definition der nachstehenden Funktion "`quadrat`"! Jeder Aufruf der Funktion soll ihren Parameter durch dessen Quadrat ersetzen. Hat also z.B. `a` den Wert 1.1, ändert ihn der Aufruf `quadrat(a)` auf 1.21.

```
quadrat (   double   p) {
p   =   p   *   p   ;    }
```

c) Zur Klasse `A` der Aufgabe 2a wird folgende Elementfunktion hinzugefügt:

```
double     div     (double p)     {
return p/2;                       }
```

Diese Funktion soll vom Hauptprogramm mit "`A::div(4)`", d.h. ohne Bezugnahme auf ein Objekt aufgerufen werden können. Tragen Sie die dafür notwendigen Änderungen in den Code ein!
Weshalb läßt sich die in Klasse `A` der Aufgabe 2a definierte Elementfunktion `add` nicht mit denselben Änderungen als "`A::add(4)`" aufrufen?

Aufgabe 4 (11 Punkte)

Eine Medienklasse sei folgendermaßen definiert:

```
class Medium {
    int sig;
    string titel;
  public:
    bool has_sig(int _sig) {
      return (sig == _sig);
    }
    Medium () {
      cin >> sig;
      titel = "";
    }
};

int main() {
  Medium m;
  cout << m.has_sig(5);
  ...
}
```

a)

b)

a) Die Medienklasse soll als Template so verallgemeinert werden, dass die Signatur auch einen anderen Typ als `int` annehmen kann. Tragen Sie oben in der rechten Spalte die dazu nötigen Änderungen an der Klassendefinition ein!

b) Geben Sie in der rechten Spalte auch an, wie das Hauptprogramm abzuändern wäre wenn die Signatur den Typ `string` haben soll.

c) Nicht jeder beliebige Datentyp kann als Typparameter für ein solches Medium verwendet werden. Geben Sie an, welche Schnittstelle (Methoden, Operationen usw.) ein Datentyp besitzen muss, damit er als Typparameter für diese Klasse eingesetzt werden kann!

Aufgabe 5 (8 Punkte)

a) Kreuzen Sie in den beiden Spalten an, welche Kombinationen der Aussagen richtig sind:

	"Überladen" sind ...	"Überschrieben" sind ...
... namensgleiche Elementfunktionen mit derselben Signatur in derselben Klasse		
... namensgleiche Elementfunktionen mit derselben Signatur in verschiedenen Klassen		
... namensgleiche Elementfunktionen mit verschiedenen Signaturen in derselben Klasse		
... namensgleiche Elementfunktionen mit verschiedenen Signaturen in verschiedenen Klassen		

b) Für die Ausgabeformatierung werden die Bezeichner `floatfield` usw. aus der Standardbibliothek verwendet.
Handelt es sich dabei um Variablen, Konstanten, Datentypen, Funktionsaufrufe oder Funktionszeiger? (Zutreffendes bitte unterstreichen)
Normalerweise steht vor diesen Bezeichnern das Präfix "`ios::`". Weshalb wird dieser Zusatz verwendet? Ist er notwendig?

c) Kreuzen Sie bitte die korrekte(n) Aussage(n) an:
Das Exception Handling wurde in C++ u.a. deshalb eingeführt
- ☐ weil es dem objektorientierten Programmierstil entspricht,
- ☐ weil es den Code der Fehlerbehandlung vom normalen Code trennt,
- ☐ damit Programme auf Ausnahmezustände des Prozessors (z.B. Adressierungsfehler) reagieren können.