# *Abschlussklausur*

**zur Vorlesung**

## »Objektorientierte Programmierung in C++«

**( 2. Semester 1999 )**

*Bitte notieren Sie auf allen (!) Lösungsblättern Ihren Namen und ihre Matrikelnummer. Die Rückgabe der Ergebnisse und die Diskussion von Beispiellösungen erfolgt am 13.7.99. Mit der vollständigen Lösung der folgenden 8(+1) Aufgaben können insgesamt 82(+2) Punkte erreicht werden. Streben Sie bei der Auswahl der von Ihnen gelösten Aufgaben 24 Punkte an!*

*Viel Erfolg !!!*

---

### 1. Europa* [12 Punkte]

Definieren Sie eine Klasse in C++, mit der sich die politische Topographie Europas darstellen lässt. Die Klasse soll eine Methode **distance** besitzen, die die minimale Zahl der (Land-)Grenzen feststellt, die zwischen zwei Hauptstädten liegt. *Zusatzaufgabe*: Definieren Sie Instanzen dieser Klasse, die die tatsächlich heute geltenden Grenzen widerspiegeln. Für je 10 richtige Grenzen mehr als falsche gibt es einen Zusatzpunkt.

---

### 2. Bitte Passend Zahlen [12 Punkte]

Schreiben Sie ein C++ -Programm, das den kleinsten Geldbetrag < 100,00 DM ermittelt, für dessen passende Bezahlung die größte Anzahl von Münzen oder Scheinen erforderlich ist.

Sei

*pmin(B):=* *die minimale Zahl von Münzen/Scheinen mit denen der Betrag B passend bezahlt werden kann*

Man ermittle also

*min { B | 0<= B <= 100 DM, pmin(B) = max { n | n=pmin(B), 0 <= B <= 100 DM } }*

mit Hilfe eines C++ -Programmes. Bemühen Sie sich dabei um einen objektorientierten Ansatz. (Münzen und Scheine sind Zahlungsmittel, eine Mark ist eine Münze, ... alle Zahlungsmittel haben einen Nominalwert, der jedoch jeweils unterschiedlich ist ...)

---

### 3. Fehlertext [10 Punkte]

Finden Sie möglichst viele Fehler in dem folgenden vermeintlichen C++-Programm. Versuchen Sie dazu, eine korrekte Implementation zu erstellen. Klassifizieren Sie die gefundenen Fehler nach

(a) Syntaxfehler (d.h. Verletzungen der Syntaxregeln der Sprache)
(b) Semantikfehler (Typ-, Schutzverletzungen, fehlende Deklarationen/Definitionen)
(c) logische Fehler (Fehler, die [nach geeigneten Korrekturen von (1) und (2)] zur Laufzeit dazu führen, dass nicht das angegebene Programmverhalten zustandekommt)
(d) memory leaks (Fehler, die [nach geeigneten Korrekturen von (1), (2) und (3)]trotz korrektem Programmverhalten zu Problemen führen können, weil dynamischer Speicher „verschwendet" wird)

---

* Idee: Martin von Löwis

**Hinweise**: Bei **setw(int)**, **setiosflags(ios::showpoint)** und **setprecision(int)** handelt es sich um sog. Manipulatoren, die, wenn sie in einen Ausgabestrom ausgegeben werden, das Format der unmittelbar nachfolgenden Zahlenausgabe bzgl. Ausgabeweite und -genauigkeit beeinflussen. Sie sind innerhalb des Standardheaderfiles **iomanip** definiert (und werden hier korrekt benutzt !) Es ist nicht festgelegt, ob **iomanip** das Headerfile **iostream** indirekt einschließt !
Die Funktion **sin** ist in übersetzter Form in geeigneten (C-) Bibliotheken verfügbar, sie verlangt ein Argument des Typs **double** und liefert ein **double**-Resultat.
Nicht alles, was auf den ersten Blick als Fehler erscheint, ist auch einer!

```
//*
Das folgende Programm soll fuer die reellen Zahlen von 0 bis 2*pi mit der
Schrittweite 0.01 die Werte der Sinusfunktion in Form einer Tabelle
ausgeben:
*/
#include <iomanip>

class header {
     static char* htext;
public
     header(){cout<<htext}
} h;

class Align {
     friend ostream& operator<<(ostream o, const Align){
          retrun o<<setw(10) /* Ausgabe in 10 Positionen */
          <<setiosflags(ios::showpoint)// immer mit Dezimalpunkt und
                                       // abschliessenden Nullen
          <<setprecision(6);//* mit 6 signifikanten Ziffern
}align;

class line : {
protected:
     Line (double arg = 0)
     {cout<align<<arg<<" | <<align<<sin(arg)<<endl;}
}

int main()
{
     for (double x=0; x!=2*PI; x+=1/100)
          new line(x);;
}

const PI=3.14159;
char* haeder:htext=" x    |   sin(x)\n";
```

---

### 4. C&D - tors [10 Punkte]

Bestimmen Sie die Ausgaben des folgenden C++ -Programms:

```
#include <iostream>

using std::cout;

void out(int i=0){i?cout<<i:cout<<'-';}
```

```
class A {
     int n;
public:
     A(int i=3):n(i){out(i);f();}
     A(const A& o){out(n=o.n);}
     ~A(){out(n);}
     virtual void f(){out(++n);}
} a;

class B : public virtual A {
     int n;
public:
     B(int i=5):n(i), A(i+1){out(i);}
     ~B(){out(n);}
     void f(){out(n--);}
} b;

class C: public virtual A {
public:
     C():A(7){out();}
     ~C(){out();}
} c;

class D: public B, public C {};

int main() {
     A * p;

     p=&a; p->f();
     p=&b; p->f();
     p=&c; p->f();

     p=new D; p->f();

     delete p;

     C* q = new D;
     q->f();
     delete q;
}
```

**Hinweis**: Die Ausgabe dieses Programmes ist zugleich die *decryption phrase* für die Entschlüsselung der vorbereiteten Beispiellösungen unter

`ftp://ftp.informatik.hu-berlin.de/pub/local/vorlesung/c++/2.klausur.99/loesungen.asc`

Auspacken mittels:

```
pgp loesungen.asc
[Enter pass phrase: ....... ]
[Should 'loesungen' be renamed to 'loesungen.gz' (Y/n)? n ]
tar xvzf loesungen
```

## 5. Traces [8 Punkte]

Implementieren Sie eine Klasse **`Tracer`**, die es erlaubt, jedes Betreten und Verlassen von Funktionskörpern (auch wenn die Funktion z.B. durch Auftreten einer Exception verlassen wird) bei einem Programmlauf durch Einschleusen eines geeigneten Makros **`TRACE`** zu verfolgen:

```
#include "tracer.h"
void g(int i) {
     TRACE(g);
     if (!i) throw 1;
}

void f() {
     TRACE(f);
     g(0);
}

int main() {
     try {
          TRACE(main);
          g(1);
          f();
     }
     catch(int) {
          TRACE(int exception);
     }
}
```

soll z.B. folgendes Protokoll erzeugen:

```
main betreten
        g betreten
        g verlassen
        f betreten
                g betreten
                g verlassen
        f verlassen
main verlassen
int exception betreten
int exception verlassen
```

**Hinweis:** Mit Hilfe von `#` (stringization) kann eine Argument eines Präprozessormakros in einen String umgewandelt werden.

```
// z.B.
#define STRING(X) #X
```

## 6. Brüche [10 Punkte]

Implementieren Sie eine Klasse **`Bruch`**, die zur Darstellung von rationalen Zahlen und ihrer Arithmetik benutzt werden kann. Versuchen Sie dabei, die ganzen Zahlen **`int`** kanonisch einzubetten:

- eine rationale Zahl kann auch eine ganze sein: z.B. **`Bruch eins = 1;`**
- Rechenoperationen **`+`**, **`-`**, **`*`**, **`/`** sollen sowohl zwischen rationalen Zahlen als auch mit ganzen Zahlen möglich sein: z.B.
  ```
  Bruch einHalb (/*Zaehler*/ 1, /*Nenner*/ 2);
  Bruch einEinHalb = einHalb + 1;
  ```
- Ist ein Konstruktor **`Bruch (double)`** sinnvoll ?
- Sollte man **`operator==`** und **`operator=`** neu definieren ?

## 7. static initialization order fiasco** [10 Punkte]

Gegeben seine die folgenden C++ -Quelltexte, sowie ein zugehöriges **`makefile`**. **`Counter`** soll dabei einen einfachen Zähler implementieren, der bei jedem Lesezugriff automatisch erhöht wird. Außerdem sollen alle **`A`**-Objekte neben der Nummerierung auch noch in einer Liste erfasst werden, die zu jedem Zeitpunkt die Ausgabe aller existierenden **`A`**-Objekte per **`printall`** erlaubt.

---

** so bezeichnet von Marshall Cline, Coautor des Buches
"C++ FAQ" by Cline and Lomow, Addison-Wesley, 1995, ISBN 0-201-58958-3.

```
// Header a.h:
#ifndef _A_H_
#define _A_H_
#include <iostream>
#include "c.h"

class A {
     int nr;
     A* next;
     static A* allA;
public:
     A():nr(Counter::count), next(allA){ allA=this; }
     friend std::ostream& operator<<(std::ostream& o, const A& a)
     { return o<<"A nr."<<a.nr<<std::endl; }
     static void printall()
     { for (A* p=allA; p!=0; p=p->next) std::cout<<*p<<std::endl; }
};
#endif
```

```
# makefile
all:
     g++ -o m a.cc someAs.cc \
         main.cc c.cc moreAs.cc
```

```
// Implementation c.cc:

#include "a.h"

A* A::allA=0;
```

```
// Header c.h:
#ifndef _C_H_
#define _C_H_

#include <iostream>

class Counter {
     int c;
     Counter():c(1){}
public:
     static Counter count;
     operator int (){return c++;}
};
#endif
```

```
// Implementation c.cc:
#include "c.h"

Counter Counter::count;
```

```
// Implementation someAs.cc:
#include "a.h"
A a1; // A nr.1 ??
A a2; // A nr.2 ??
```

```
// Implementation moreAs.cc:
#include "a.h"
A a3; // A nr.3 ??
A a4; // A nr.4 ??
```

```
// Implementation main.cc:
#include "c.h"                    // beim Aufruf von printall sollten
#include "a.h"                    // 4 A-Objekte mit den Nummern
                                  // 1 .. 4 existieren
int main() { A::printall(); }
```

Die Ausführung des resultierenden Programms liefert jedoch:

```
A nr.2
A nr.1
A nr.1
A nr.0
```

und zeigt, dass offenbar irgend etwas mit dem Zähler nicht stimmt. Erklären Sie den aufgetretenen Effekt **[4 Punkte]** und verändern Sie (nur!) die Quelltexte `c.h` und `c.cc` so, dass der Fehler behoben wird **[6 Punkte]**.

## 8. Pascal [10 Punkte]

Definieren Sie C++ -Klassen, deren Instanzen Zeilen des Pascal'schen Dreiecks repräsentieren:

```
          1
        1 1
      1 2 1
    1 3 3 1
  1 4 6 4 1
1 5 10 10 5 1
```

Geben Sie zwei verschiedene Implementationen an:

eine, bei der die Werte des jeweiligen Zeilenvektors bereits von Compiler ermittelt werden:

```
Pascal<1> p1;
Pascal<2> p2;
Pascal<3> p3;
Pascal<4> p4;
Pascal<5> p5;
Pascal<6> p6;
```

und eine, bei der dies erst zur Laufzeit geschieht:

```
Pascal p1(1);
Pascal p2(2);
Pascal p3(3);
Pascal p4(4);
Pascal p5(5);
Pascal p6(6);
```

In jedem Fall soll

```
std::cout << p1 << p2 << p3 << p4 << p5 << p6;
```

das obige Dreieck (ohne Zentrierung) erzeugen. Vergleichen Sie beide Varianten.

## 9. Zugabe [2 Punkte]

Welches Problem verbirgt sich in dem folgenden C++ -Programm ? Es sollte eigentlich 42 ausgeben, gibt aber gar nichts aus!

```
// Copyright © 1996, Gimpel Software : bug752.cc
#include <iostream>

int sum = 0;
class Add
{
public:
     Add( int a, int b, int c )  { sum += a; Add x(b,c); }
          Add( int a, int b )    { sum += a;   Add x(b); }
     Add( int a )                { sum += a;    Add x(); }
     Add()                       { std::cout << sum;     }
};

int main()
{
     Add x( 21, 14, 7 );
     return 0;
}
```