

Mündliche Prüfungen für Informatik III sind am Montag, den 28. Januar ab 8:30 Uhr zusammen mit Prof. Isernhagen geplant.

Insgesamt ist die Klausur somit ganz gut ausgefallen (Schnitt 3,4, ohne Durchfaller sogar 2,6),

- a.) 10 mal sehr gut,
- b.) 20 mal gut,
- c.) 13 mal befriedigend,
- d.) 15 mal ausreichend,
- e.) 30 mangelhaft (von 88).

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

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

Klausur im WS 2001/02 :

Informatik III — Lösungen —

Hilfsmittel sind bis auf Computer, Handy etc. erlaubt !

Bitte Aufgabenblätter mit abgeben !

Die Lösungen sind in den meisten Fällen auf einem separaten Aufgabenblatt anzugeben. Falls der Platz auf dem Aufgabenblatt es zulässt, können die Lösungen auch dort notiert werden.

Bitte notieren Sie auf **allen** Aufgabenblättern und separaten Blättern Ihren Namen.

Auf eine absolut korrekte Anzahl der Blanks und Zeilenumbrüche braucht bei der Ausgabe nicht geachtet zu werden. Dafür werden keine Punkte abgezogen.

Es ist nicht unbedingt erforderlich, dass die Programme hocheffizient sind. Wichtiger ist, dass sie verstehbar sind. Eine gute Dokumentation ist keinesfalls *verboten*.

Hinweis: In den folgenden Programmen wird sehr häufig die globale Variable *datei* verwendet. Hierfür kann der Einfachheit halber die Variable *cout* angenommen werden. Die Variable *datei* dient lediglich bei der Klausurerstellung dem Zweck der Ausgabeumlenkung.

Aufgabe 1 : Kontrollstrukturen, Post/Pre-Increment

ca. 10 Punkte

Was gibt das folgende Programm in die Datei *datei* aus?

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

ofstream datei("test.txt", ios::out);

int main()
{
  int i(0);
  for (i=0; i<5; ++i) {
    datei << i << " ";
  }
  datei << endl;
  for (i=0; i<5; i++) {
    datei << i << " ";
  }
  datei << endl;
  for (i=0; i<5; i++) {
    datei << i++ << " ";
  }
  datei << endl;
  for (i=0; i<5; i++) {
    datei << ++i << " ";
  }
  datei << endl;
  i=0;
  for ( ; i++ < 5; ) {
    datei << i << " ";
  }
}
```

Bewertung: 2 + 2 + 2 + 2 + 2

Ausgabe:

```
0 1 2 3 4
0 1 2 3 4
0 2 4
1 3 5
1 2 3 4 5
```

Aufgabe 2 : Parameterübergabe

ca. 10 Punkte

Gegeben ist der folgende Ausschnitt aus einem Programm:

```
/*-----*/
void funk1(int i, int j);
void funk2(int& i, int* j);
void funk3(float i, int j[]);

int main()
{
  int k(14);
  int feld[40]; feld[4]=16;

  funk1(k, 14); /*1*/
  funk2(14, &k); /*2*/
  funk3(static_cast<float>(14), &k); /*3*/
  funk1(feld[4], k); /*4*/
  funk2(feld[4], feld); /*5*/
  funk3(14.3f, &k); /*6*/
}
```

Welche der Funktionsaufrufe führt zu einer Compilerfehlermeldung? Korrigieren Sie die entsprechenden Funktionsaufrufe, sodass sich weder eine Fehlermeldung noch eine Warnung ergibt. (Es geht hier **nicht** darum, ob die entsprechenden Aufrufe sinnvoll sind, sondern nur, dass sie syntaktisch in Ordnung sind.)

Nr.	O.K.	falsch	eventuelle Korrektur
1.	X		
2.		X	funk2(k, &k); /*2*/
3.	X		
4.	X		
5.	X		
6.	X		

/*2*/ ist falsch, Für einen Referenzparameter muss eine Variable übergeben werden, also z.B.

Aufgabe 3 : nachträgliches Weihnachtsgeschenk?

ca. 1 Punkte

Wie nennt man ein Programm zur Übersetzung von einer Programmiersprache in eine andere?

- A Compiler
- B Commander
- C Complainer

a.)

Es reicht aus, wenn Sie in jeder Spalte nur die Veränderungen zur vorherigen Spalte eintragen.

Variable	Adresse	/* 1 */	/* 2 */	/* 3 */	/* 4 */	/* 5 */	/* 6 */
i	1000	20	24				
j	1004	30			61		
k	1008	40					30
a1	1060	1008					
a2	1064	30					
a3	1068	1000		1004		1008	

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

ofstream datei("test.txt", ios::out);
int Verdaechtig(int* a1, int a2, int* a3)
{
    a2 = a2;           /* 1 */
    *a3 = 24;         /* 2 */
    ++a3;            /* 3 */
    *a3 = 61;         /* 4 */
    a3 = a1;         /* 5 */
    *a3 = a2;         /* 6 */
    return *(--a3);
}
/*-----*/
int main()
{
    int i = 20;
    int j = 30;
    int k = 40;
    /* .. */
    datei << Verdaechtig(&k, j, &i) << endl;
    datei << i << " " << j << " " << k << endl;
    return 0;
}
```

b.)

Was bedeutet die Anweisung *(--a3) (return-Anweisung von Verdaechtig) in Funktions-

D Composer

A. Compiler

Aufgabe 4 : Zeiger, Adressen

ca. 6+2+3+2 Punkte

Gegeben ist der untenstehende Ausschnitt aus einem Programm:

Geben Sie die Speicherbelegung der Variablen i, j, k, a1, a2, a3 zu den im Programm mit /* 1 */, /* 2 */, /* 3 */, /* 4 */, /* 5 */, /* 6 */ gekennzeichneten Zeitpunkten an.

Gehen Sie dabei davon aus, dass der Speicher auf dem Stack von den niedrigen zu den hohen Adresse wächst, dass also wie gezeigt, die Variable i auf dem Stack eine niedrigere Adresse wie die später angelegte Variable j belegt. Außerdem soll jeder Pointer 4 Bytes im Speicher belegen, d.h. sizeof(void*) ergäbe 4.

Schreibweise (a - b; bedeutet in Funktions-Schreibweise operator-(a, b)? Die Lösung dieser Aufgabe könnte auch für die Lösung von Teil a) sehr hilfreich sein.

```
operator*(operator--(a3));
```

c.)

Was gibt das main-Programm in die Datei datei aus?

61

24 61 30

2 Punkte für Funktionswert, 1 Punkt für Rest

d.)

Was gibt das Programm in die Datei datei aus, wenn die letzte Anweisung der Funktion Verdaechtig geändert wird in: Was gibt das main-Programm in die Datei datei aus?

```
return *(a3--); /* statt return *(--a3); */
```

30

24 61 30

1,5 Punkte für Funktionswert, 0,5 für Rest

Aufgabe 5 : STL, Konstruktoren, Destruktoren, Polymorphie

ca. 4+4+4+7+9+3 Punkte

Gegeben ist das folgende Hauptprogramm:

```
#include <fstream>
#include <list>
#include <algorithm>
using namespace std;

ofstream datei("test.txt", ios::out);

/* Der Teil ab hier wird bei den
   folgenden Aufgaben vorgestellt. */

int main()
{
    // Aufgabe a
    datei << "funk1\n"; funk1(); datei << endl;
    // Aufgabe b
    datei << "\nfunk2\n"; funk2(); datei << endl;
    // Aufgabe c
    datei << "\nfunk3\n"; funk3(); datei << endl;
    // Aufgabe d und e und f
    datei << "\nfunk4\n"; funk4(); datei << endl;

    return 0;
}
```

a)

Was gibt der Aufruf der folgenden Funktion `funk1` in die Datei `datei` aus. Als Hilfe sind im Anhang zur Wiederholung nochmals die Definition des STL-Algorithmus `for_each` sowie ein Ausschnitt aus dem Destruktor von der STL-Klasse `list` angegeben.

```
void WPrint(int i) {datei << i << " ";}
void PPrint(int* pi){datei << *pi << " "};
```

```
void funk1()
{
    typedef list<int> T_IntList;
    typedef list<int*> T_PIntList;

    T_IntList wListe;
    T_PIntList pListe;

    int i1(1), i2(2), i3(3);

    wListe.push_back(i1+10);
    wListe.push_back(i2+10);
    wListe.push_back(i3+10);

    pListe.push_back(&i1);
    pListe.push_back(&i2);
    pListe.push_back(&i3);

    for_each(++wListe.begin(), wListe.end()--,
             WPrint);
    datei << endl;
    for_each(pListe.begin(), pListe.end(),
             PPrint);
}
```

12 13

1 2 3

Beginn bei 12: 1

Ende bei 13 : 1

Beginn bei 1: 1

Ende bei 3 : 1

b)

```
/******
class Zahl
{
public:
    Zahl(int w);
    virtual ~Zahl();
    void PrType() const;
    virtual void PrWert() const;
protected:
    int wert;
};

Zahl::Zahl(int w): wert(w) {
    datei << "+Z" << wert << " ";
}
Zahl::~~Zahl() {
    datei << "-Z" << wert << " ";
}

/******
void funk2()
{
    list<Zahl> zListe;
    Zahl z1(1), z2(2);
    zListe.push_back(z1);
    zListe.push_back(z2);
}
```

Was gibt der Aufruf der Funktion `funk2` in die Datei `datei` aus.

```
+Z1 +Z2 -Z2 -Z1 -Z1 -Z2
+Z1 Z2:                               1
Copy Konstruktor gibt nichts aus: 1
Destruktoraufrufe:                    1
Reihenfolge von Destruktoraufrufen, wenn
    Copy Konstruktor weggelassen wurde: +1
-Z2 -Z1:                               1
```

c)

```
/******
void funk3()
{
    list<Zahl*> zpListe;
    Zahl z1(1);
    zpListe.push_back(&z1);
    zpListe.push_back(&z1);
    zpListe.push_back(&z1);
}
```

Was gibt der Aufruf der Funktion `funk3` in die Datei `datei` aus.

```
+Z1 -Z1
+Z1:                                     1
Copy-Konstruktor sowie
    Destruktoren geben nichts aus: 2
-Z1:                                     1
```

d)

```

/*****/
void Zahl::PrType() const {
    datei << "ZAHL" << " ";
}
void Zahl::PrWert() const {
    datei << wert << " ";
}

class Real : public Zahl
{
public:
Real(int w);
virtual ~Real();
void PrType() const;
virtual void PrWert() const;
};

Real::Real(int w): Zahl(w) {
datei << "+R" << wert << " ";
}
Real::~Real() {
datei << "-R" << wert << " ";
}

void Real::PrType() const {
datei << "REAL " << " ";
}
void Real::PrWert() const {
datei << wert << ".0 ";
}

void ZPrint(Zahl* pz){
pz->PrType(); pz->PrWert();
}

void ZPrintConv(Zahl* az){
Zahl z(*az);
Zahl* pz = &z;
pz->PrType(); pz->PrWert();
}

// siehe Anhang fuer for_each
void funk4()
{
    list<Zahl*> zpListe;
    Zahl z1(1);
    Real* pr2 = new Real(2);
    Real r3(3);
    Zahl* pz=&r3;

    zpListe.push_back(&z1);
    zpListe.push_back(pr2);
    zpListe.push_back( pz);

#ifdef WITH_FOREACH
    datei << endl;
    for_each(zpListe.begin(),zpListe.end(),
            ZPrint);
    datei << endl;
    for_each(zpListe.begin(),zpListe.end(),
            ZPrintConv);
    datei << endl;
#endif /* WITH_FOREACH */
}

```

Was gibt der Aufruf der Funktion funk4 in die Da-

tei datei aus, wenn man davon ausgeht, dass das Programm ohne die Makrodefinition WITH_FOREACH übersetzt wurde, d.h. die Zeilen zwischen den Präprozessorbefehlen #ifdef WITH_FOREACH und #endif nicht existieren würden.

Tipp: Gehen Sie Anweisung für Anweisung durch.

```
+Z1 +Z2 +R2 +Z3 +R3
-R3 -Z3 -Z1
```

```
+Z1: 0,5; + Z2 + R2: 1
+Z3 +R3: 1, pz macht nicht: 1
push_back keine Ausgabe: 1
-Z3 -R3 : 1, nicht bei -Z2, -R2: 1
Z1: 0,5
```

e)

Was gibt der Aufruf der Funktion funk4 **zusätzlich** in die Datei datei aus, wenn das Programm mit der Makrodefinition WITH_FOREACH übersetzt wurde, d.h. die beiden Zeilen mit den Präprozessorbefehlen #ifdef WITH_FOREACH und #endif selbst nicht existieren würden.

Tipp: Gehen Sie Anweisung für Anweisung durch.

```
ZAHL 1 ZAHL 2.0 ZAHL 3.0
ZAHL 1 -Z1 ZAHL 2 -Z2 ZAHL 3 -Z3
```

```
ZAHL 1: 1
ZAHL 2.0 ZAHL 3.0: 3
Immer ZAHL: 1;
immer Zahl-Wert: 1,5;
Kein Ausgabe des Copy-Konstruktors: 1
Destruktorausgaben in ZprintConv: 1,5
```

f)

Wo treten durch Aufruf von Funktion funk4 Speicherlecks auf? Korrigieren Sie das Programm daraufhin.

In funk4 fehlt eine Freigabe des Speichers, auf den pr2 verweist:

```
delete pr2;
1,5 + 1,5 Pkte
```

Aufgabe 6 : Zeigersemantik

ca. 15+2 Punkte

Implementieren Sie für die folgende Klasse *Schlange* einen Destruktor einen Copy-Konstruktor und einen Zuweisungsoperator.

Hinweis: Sie können Destruktor, Copy-Konstruktor und Zuweisungsoperator dieser drei Teilaufgaben inline implementieren, d.h. es wird angenommen, dass der von Ihnen hier implementierte Code im public-Teil nach dem Konstruktor von Schlange folgt.

```
class Schlange { /* FIFO*/
    int *Data;
    int maxsize;
    int n;
public:
    Schlange(int siz=20)
    { Data = new int[maxsize=siz]; n = 0; }

```

```

/* hier sollte Ihr Code des
   Destruktors etc. stehen */

void Einfuegen(int x) { Data[n++] = x; }
int Groesse()      { return n; }
int GibErstes() {
    int wert = Data[0]; --n;
    for (int i=0; i<n; ++i) {
        Data[i]=Data[i+1];
    }
    return wert;
}
};

Nach Ihrer Implementierung soll das folgende Programm lauffähig sein, ohne Speicherlecks zu hinterlassen. Die Ausgabe des Programm soll dann 1 2 3 4 und 1 2 3 4 sein.

int main(){
Schlange s1;
s1.Einfuegen(1);
s1.Einfuegen(2);
s1.Einfuegen(3);
s1.Einfuegen(4);

Schlange s2(s1), s3(44);
datei << s2.GibErstes() << " ";

s2 = s3 = s1 = s1;
datei << s2.GibErstes() << " ";
return 0;
}

~Schlange()      { delete[] Data; }

Schlange(const Schlange& s2):
    maxsize(s2.maxsize), n(s2.n)
{
    Data = new int[maxsize];
    for (int i=0; i<maxsize; ++i){
        Data[i] = s2.Data[i];
    }
}

Schlange& operator=(const Schlange& s2) {
    if (this != &s2){
        delete [] Data; n=s2.n;
        maxsize = s2.maxsize;
        Data = new int [maxsize];
        for (int i=0; i<maxsize; ++i){
            Data[i] = s2.Data[i];
        }
    }
}

return *this;
}

Destruktor:                2

Schnittstelle des Copy-Konstruktors: 1
Zuweisung an n, maxsize:    1

```

```

Zuweisung an Data mit new      1
Kein delete in Konstruktor, wenn new 1
For-Schleife                    1

Schnittstelle operator=        1
Eigenzuweisung verhindern     0,5
Zuweisung an n, maxsize:      1
Delete-Aufruf                  1
Zuweisung an Data mit new      1
For-Schleife                    1
return *this                    0,5

```

b)

Welche der Methoden und Elemente der Klasse **Schlange** (mit Ihren zusätzlich implementierten) können als konstant implementiert werden.

Kein Element kann konstant implementiert werden, denn alle Elemente werden im Zuweisungsoperator modifiziert. Nur die Methode **Groesse** kann und sollte als konstante Methode implementiert werden.

```

int Groesse() const { return n; }
1 für Methode, 1 für Elemente,
+1, wenn Angabe warum

```

Aufgabe 7 : Templates

ca. 17+3 Punkte

Schreiben Sie eine Klasse **UeberwZahl**, die es ermöglicht, dass einer Variablen nur ganz bestimmte Werte aus einem Intervall zugewiesen werden können. Falls versucht wird, andere Werte zuzuweisen, bleiben die bisherigen Werte erhalten.

Das folgende Programm zeigt eine beispielhafte Anwendung der Klasse, die in der Datei **UeberwZahl.h** definiert wird. Ihre Methoden werden alle in der Datei **UeberwZahl.cxx** definiert.

Vergessen Sie nicht, die notwendigen **include**-Anweisungen und einen **Include-Wächter** zu implementieren. Achten Sie auch darauf, soweit wie möglich das Schlüsselwort **const** zu verwenden.

```

#include <fstream>
using namespace std;
#include "UeberwZahl.h"

ofstream datei("test.txt", ios::out);

int main()
{
    // Wert 14, wert aus [0..22]
    UeberwZahl<int> wert(14, 0, 22);
    // Wert 14.4, d aus [-1..100]
    UeberwZahl<double> d(14.4, -1, 100);

    wert = 61;
    wert = 17;
    wert = -1;

    d = 11.0;

    datei << "*** " << wert << " ***" << endl;

```

```

datei << "*** " << d    << " ***" << endl;

datei << "*** wert:" << wert.GetWert()
    << " *** " << endl;
datei << "*** d    :" << d.GetWert()
    << " *** " << endl;
return 0;
}

```

Dieses Programm könnte z.B. die folgende Ausgabe erzeugen:

```

Ueberwachung von 14 in [0..22]
Ueberwachung von 14.4 in [-1..100]
Neuer Wert falsch 61 in [0..22]
Alter Wert 14 bleibt.
Neuer Wert OK 17 in [0..22]
Neuer Wert falsch -1 in [0..22]
Alter Wert 17 bleibt.
Neuer Wert OK 11 in [-1..100]
*** 17 in [0..22] ***
*** 11 in [-1..100] ***
*** wert:17 ***
*** d    :11 ***

```

In Ihrer Implementierung brauchen die Methoden der Klasse selbst keine Ausgaben zu erzeugen. Die angegebenen Ausgaben, dienen lediglich zur Veranschaulichung der Funktionalität.

a.)

Es ist ausreichend, wenn obiges Programm übersetzt werden kann und die entsprechende Funktionalität (ohne Ausgaben) aufweist. Zusätzliche Methoden der Klasse, die eventuell für weitere Anwendungen sinnvoll sein könnten, sollen **nicht** implementiert werden, d.h. implementieren Sie die erforderlichen Elemente der Klasse, einen geeigneten Konstruktor, einen geeigneten Operator (Welchen?) und die Methode `GetWert`.

b.)

Falls man die Klasse `UeberwZahl` für beliebige Typen, nicht nur für Standardtypen, verwenden will, welche Eigenschaften müssen diese Typen dann mindestens aufweisen? (Welche Methoden etc. muss es für sie geben, auch wenn diese automatisch vom C++-System erzeugt werden sollten?)

```

// Datei UeberZahl.h
#ifndef UEBERWZAHL_H
#define UEBERWZAHL_H

#include <fstream>
using namespace std;
extern ofstream datei;

template <class T>
class UeberwZahl
{
public:
    UeberwZahl(const T& w, const T& amin,
               const T& amax);
    UeberwZahl& operator=(const T& w);
    const T& GetWert() const;
    void TestOutput(const char*, const T& w,
                   ostream& d) const;

```

```

private:
    T wert;
    const T min;
    const T max;
};

template <class T>
ostream& operator<<(ostream& str,
                  const UeberwZahl<T>& w);

#endif

/*****
// Datei UeberZahl.cpp
*****/
#include "UeberwZahl.h"

template <class T>
UeberwZahl<T>::UeberwZahl
(const T& w, const T& amin, const T& amax):
    wert(w), min(amin), max(amax)
{
    TestOutput("Ueberwachung von ", wert, datei);
    datei << endl;
}

template <class T>
UeberwZahl<T>& UeberwZahl<T>::operator=(const T& w)
{
    if ((w >= min) && (w <= max))
    {
        wert = w;
        TestOutput("Neuer Wert OK ", wert, datei);
        datei << endl;
    }
    else
    {
        TestOutput("Neuer Wert falsch ", w,datei);
        datei << "\nAlter Wert " << wert
            << " bleibt." << endl;
    }
}

return *this;
}

template <class T>
void UeberwZahl<T>::TestOutput
(const char* text, const T& w, ostream& d) const
{
    d << text << w
    << " in [" << min << ".."
    << max << "];"
}

template <class T>
const T& UeberwZahl<T>::GetWert() const
{
    return wert;
}

template <class T>
ostream& operator<<
(ostream& str, const UeberwZahl<T>& w)
{
    w.TestOutput(" ", w.GetWert(), str);
    return str;
}

```

}

Punktevergabe: 17+3

Header-Datei

#####

Wächter:	1,5
template <class >	1
public, private-Teil	1
Konstruktor-Schnittstelle	1,5
Operator=	1,5
GetWert	1
Member der Klasse	1,5
Wenn min und max const	+2
#endif beim Wächter	0,5

Source-Datei:

#####

Include von Header:	1
Template Konstruktor:	0,5
Schnittstelle Konstruktor	0,5
Zuweisungen im Konstr.	1
Zuweisungen nach Doppelpunkt	
Mit const	+1
Template operator=	0,5
Schnittstelle	0,5
If mit Anweisungen	2
GetWert	1,5

b.)

Copy-Konstruktor, Zuweisungs-operator Destruktor,
operator<=, operator>= oder andere

Anhang 1: Destruktor von list

```
template <class T>
list<T>::~~list(){
    //...
    erase(begin(), end());
    //...
}
```

Anhang 2: Schablonenfunktion for_each

```
template<class It, class Fn> inline
Fn for_each(It F, It L, Fn Op) {
    for (; F != L; ++F)
        Op(*F);
    return (Op);
}
```

Anhang 3: Geplante Punktevergabe

Punktziel	Sonderpunkte	erreicht
A1: 10 P		
A2: 10 P		
A3: 1 P		
A4: 15 P		
A5: 31 P		
A6: 15 P		
A7: 20 P		
Summe 102 P		