

Name:

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

Matrikelnummer:

Punktzahl:

Ergebnis:

Freiversuch

F1

F2

F3

Klausur im WS 2004/05 :

Informatik III — Lösungen —

Hilfsmittel sind bis auf Computer, Handy etc. erlaubt !

Bitte Aufgabenblätter mit abgeben !

Austausch von Hilfsmitteln mit Kommilitonen ist **nicht** erlaubt !

Die Lösungen können in einigen Fällen hier auf dem Aufgabenblatt angegeben werden. Sie dürfen aber auch Ihre Lösungen, falls erforderlich, auf separaten Aufgabenblättern notieren.

Bitte notieren Sie auf **allen** Aufgabenblättern und separaten 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 Programmen wird sehr häufig die globale Variable *datei* verwendet. Hierfür kann der Einfachheit halber die Variable *cout* angenommen werden. Die Variable *datei* diene lediglich bei der Klausurerstellung dem Zweck der Ausgabeumlenkung.

Geplante Punktevergabe

Punktziel	Im einzelnen	
A1: 17 P.	(10+7)	
A2: 33 P.	(21+12)	
A3: 12 P.		
A4: 11 P.		
A5: 14 P.		
A6: 13 P.		
Summe 100 P.		

Aufgabe 1 : Methodenimplementierung

ca. 17 Punkte

a.) Implementieren Sie die Schnittstelle (Header-Datei) der Klasse `Dokument` mit den Attributen

- `seiten` (für die Anzahl der Seiten des Dokuments) und
- `datum` als `string` (für das Erstellungsdatum des Dokuments).

Außerdem soll die Klasse über die folgenden Methoden verfügen:

- Standard-Konstruktor,

b.) Implementieren Sie nun für diese Klasse

- Standard-Konstruktor,

- Kopier-Konstruktor,
- Zuweisungsoperator,
- `SetzeSeitenzahl`, um die Anzahl der Seiten auf einen bestimmten Wert zu setzen, und
- `IstDokumentAelterAls`. Die Methode liefert `true`, falls das Objekt älter als das übergebene Argument der Methode ist.

Es geht also lediglich um die Header-Datei, nicht um die Implementierung der Methoden selbst.

- Kopier-Konstruktor und
- Zuweisungsoperator.

Lösung:Listing 1: (`./Code1/Dokument.h`)

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

// globale Variable fuer Ausgabedatei
// kann entsprechend cout verwendet werden
extern ofstream datei;

class Dokument{
    int seiten;
    string datum;
public:
    Dokument();
    Dokument(const Dokument& d);
    Dokument& operator=(const Dokument& d);
    void SetzeSeitenzahl(int s);
    bool IstDokumentAelterAls(const Dokument& d) const;
};
```

Listing 2: (`./Code1/Dokument.cxx`)

```
#include "Dokument.h"

Dokument::Dokument() {
    seiten=0;
    datum="";
}

Dokument::Dokument(const Dokument& d) {
    seiten=d.seiten;
    datum=d.datum;
}

Dokument& Dokument::operator=(const Dokument& d) {
    seiten=d.seiten;
    datum=d.datum;
    return *this;
}
```

Aufgabe 2 : Konstruktoren und Destruktoren

ca. 33 Punkte

Gegeben ist die Klasse Person

Listing 3: (./Code1/Person.h)

```

#ifndef Person_H
#define Person_H
// globale Variable fuer Ausgabedatei
// kann entsprechend cout verwendet werden
extern ofstream datei;
#include <string>
using namespace std;

class Person {
    int gr; // Groesse in cm
    public:
        Person(): gr(0) {datei << "␣P␣" << gr;}
        Person(int g): gr(g) {datei << "␣P␣" << gr;}
        ~Person() {datei << "␣P␣" << gr;}

        // wird erst im 2. Teil gebraucht
        string Intelligent() {return "␣nein␣";}
        virtual string Typ() {return "␣P␣";}
};
#endif

```

sowie die Klasse Student, siehe unten.

Im Folgenden können Sie Ihre Ausgaben im Code (am rechten Rand) notieren.

- Welche Ausgabe erzeugt der Aufruf von der Funktion `funk1`?
- Welche Ausgabe erzeugt der Aufruf von der Funktion `funk2`?
- Welche Ausgabe erzeugt der Aufruf von der Funktion `funk3`?

Listing 4: (./Code1/funk13.cxx)

```

#include <fstream>
using namespace std;
#include "Person.h"
// globale Variable fuer Ausgabedatei
// kann entsprechend cout verwendet werden
extern ofstream datei;

void funk1() {
    Person p1;
    Person p2(21);
}

void funk2() {
    Person* p=NULL;
    Person* feld[3];
    feld[1] = p;
    datei << "Ende" << endl;
}

```

```

void funk3()
{
    Person* p1 = new Person(11);
    Person* p2 = new Person(21);
    datei << "␣Zuweisung:␣";

    p1 = p2;
    datei << "␣2.␣Zuweisung:␣\n";
    *p1 = *p2;
    datei << "Ende" << endl;
}

```

Listing 5: (./Code1/Student.h)

```

#ifndef Student_H
#define Student_H
#include "Person.h"

class Student: public Person {
    string fach;
    public:
        Student(): Person(180), fach("Info") {
            datei << "␣S␣" << fach;}
        Student(string f) {
            fach = f; datei << "␣+S␣" << fach;}
        ~Student() {datei << "␣-S␣" << fach;}

        // wird erst im 2. Teil gebraucht
        string Intelligent() {return "␣ja␣";}
        virtual string Typ() {return "␣S␣";}
};
#endif

```

- Welche Ausgabe erzeugt der Aufruf von der Funktion `funk4`?
- Welche Ausgabe erzeugt der Aufruf von der Funktion `funk5`?

Listing 6: (./Code1/funk45.cxx)

```

#include <fstream>
#include <memory>
using namespace std;
#include "Student.h"
extern ofstream datei; // s.o

void Test(Person pers){
    datei << "\nTest";
}

void funk4()
{
    Student s1("Mathe");
    Student s2;

    Test(s2);

    Test(s1);
    datei << "\nEnde␣funk4\n";
}

```

```
void funk5()
{
    Person* p1 = new Person(11);
    auto_ptr<Person> ap1(new Person(21));
    Person* p3 = new Person(31);

    auto_ptr<Person> ap2(p1);
    ap2 = ap1;
    datei << "Ende" << endl;
}
```

Um Ihnen das Umblättern zu ersparen, nochmals der Code der beiden Klassendefinitionen:

Listing 7: (./Code1/Person.h)

```
#ifndef Person_H
#define Person_H
// globale Variable fuer Ausgabedatei
// kann entsprechend cout verwendet werden
extern ofstream datei;
#include <string>
using namespace std;

class Person {
    int gr; // Groesse in cm
public:
    Person(): gr(0) {datei << "_+P_" << gr;}
    Person(int g): gr(g) {datei << "_+P_" << gr;}
    ~Person() {datei << "_-P_" << gr;}

    // wird erst im 2. Teil gebraucht
    string Intelligent() {return "_nein_";}
    virtual string Typ() {return "_P_";}
};
#endif
```

Listing 8: (./Code1/Student.h)

```
#ifndef Student_H
#define Student_H
#include "Person.h"

class Student: public Person {
    string fach;
public:
    Student(): Person(180), fach("Info") {
        datei << "_+S_" << fach;}
    Student(string f) {
        fach = f; datei << "_+S_" << fach;}
    ~Student() {datei << "_-S_" << fach;}

    // wird erst im 2. Teil gebraucht
    string Intelligent() {return "_ja_";}
    virtual string Typ() {return "_S_";}
};
#endif
```

Lösung:

Bewertung: a): 21+12

Listing 9: (./Code1/ausgabe2.txt)

```
funk1:
+P 0 +P 21 -P 21 -P 0
funk2
Ende
funk3
+P 11 +P 21 Zuweisung: 2. Zuweisung:
Ende
funk4
+P 0 +S Mathe +P 180 +S Info
Test -P 180
Test -P 0
Ende funk4
-S Info -P 180 -S Mathe -P 0
funk5
+P 11 +P 21 +P 31 -P 11Ende
-P 21
Nach funk1_5
+P 180 +S Info +P 0
Vor funk6_9
#####

funk6
S P
ja nein
funk7
S P
ja nein
-S Info -P 180 -P 0funk8
S P
nein nein
funk9
P P
nein nein
-P 180 -P 0 -S Info -P 180 -P 0
#####
-P 0 -S Info -P 180
```

In den folgenden vier Teilaufgaben sollen die Ausgaben der Konstruktoren und Destruktoren ignoriert werden!!!

- Welche Ausgabe erzeugt der Aufruf von der Funktion funk6?
 - Welche Ausgabe erzeugt der Aufruf von der Funktion funk7?
 - Welche Ausgabe erzeugt der Aufruf von der Funktion funk8?
 - Welche Ausgabe erzeugt der Aufruf von der Funktion funk9?
- Nehmen Sie an, dass die 4 Funktionen entsprechend Funktion funk6_9 mit einer Instanz von Student und einer von Person aufgerufen werden.

Listing 10: (./Code1/funk69.cxx)

```
#include <fstream>
using namespace std;
#include "Person.h"
#include "Student.h"
// globale Variable fuer Ausgabedatei wie cout
extern ofstream datei;

void funk6(Student& s1, Person& p1)
{
```

```
void funk7(Student s1, Person p1)
{
    datei << s1.Typ() << p1.Typ() << endl;
    datei<<s1.Intelligent()<<p1.Intelligent()<<endl;
}
```

```
datei
<<
s1
Typ
() <<
p1
Typ
() <<
endl
,
datei<<s1.Intelligent()<<p1.Intelligent()<<endl;
}
```

```
void funk8(Person& p1, Person& p2)
{
    datei << p1.Typ() << p2.Typ() << endl;
    datei<<p1.Intelligent()<<p2.Intelligent()<<endl;
}
```

```
void funk9(Person p1, Person p2)
{
    datei << p1.Typ() << p2.Typ() << endl;
    datei<<p1.Intelligent()<<p2.Intelligent()<<endl;
}
```

```
void funk6_9(Student s1, Person p1) {
    datei << "\nfunk6" << endl;
        funk6(s1, p1);
    datei << "funk7" << endl;
        funk7(s1, p1);
    datei << "funk8" << endl;
        funk8(s1, p1);
    datei << "funk9" << endl;
        funk9(s1, p1);
}
```

Aufgabe 3 : Textfragen agile Software-Entwicklung

ca. 12 Punkte

- a.) Nennen Sie neben Kunde vor Ort und gemeinsame Verantwortung fünf (nicht mehr) weitere Basistechniken des eXtreme Programming. **Lösung:**
40-Stunden-Woche, Metapher, 2-Leute-1-Bildschirm, Gemeinsame Verantwortung, Testen, Programmierstandard usw.
- b.) Erklären Sie mit 2-3 Sätzen die eXtreme Programming Basistechnik gemeinsame Verantwortung. **Lösung:**
Bei gemeinsamer Verantwortung gehört der gesamte Code nicht einem einzelnen, sondern dem Team. Jeder darf damit jederzeit jeden Code ändern.
- c.) Bei der Planung von Software-Projekten spielen neben den Kosten drei weitere Variablen eine Rolle. Welche? **Lösung:**
Qualität, Umfang, Zeit
- d.) Was besagt Brooks Gesetz? **Lösung:**
Adding men to a late project makes it later.
- e.) Erklären Sie den Begriff idealer Tag. Erklären Sie an einem Zahlenbeispiel den Begriff des Load Factors. **Lösung:**
Bei einem idealen Tag wird angenommen, dass ein Programmierer seine gesamte Arbeitszeit für die geplanten Programmieraufgaben verwendet, er also nicht mit anderem (Telefonieren, Unterstützung von Team-Kollegen usw.) beschäftigt ist.
Nehmen wir an während 10 Arbeitstagen hat ein Programmierer geplante Aufgaben im Umfang von 3 idealen Tagen erledigt. Dann beträgt sein Load-Factor $3/10 = 0,3$ also 30%.

Aufgabe 4 : Textfragen zu C++

ca. 11 Punkte

- a.) Wofür steht die Abkürzung *STL*? **Lösung:**
Standard Template Library
- b.) Was ist bei der Parameterübergabe einer Instanz der Klasse `auto_ptr` an eine Funktion zu beachten? **Lösung:**
`auto_ptr` sind als Referenzparameter an Funktionen zu übergeben.
- c.) Können Destruktoren Argumente haben? **Lösung:**
nein
- d.) Geben Sie ein Codebeispiel für einen Umwandlungskonstruktor (z.B. Klassenschnittstelle mit Inline-Code). **Lösung:**

```
class A {
    int a;
    public:
    A(int par) {a=par;}
};
```

- e.) Welche Methoden etc. sind erforderlich, damit eine Klasse über eine minimale Standardschnittstelle verfügt? **Lösung:**
Standard-Konstruktor, Destruktor, Zuweisungsoperator und Kopierkonstruktor
- f.) Wie können Sie verhindern, dass bei einer Klasse ohne explizit implementierten Kopierkonstruktor dieser versehentlich aufgerufen wird? **Lösung:**
Die entsprechende Methode wird im `private`-Teil der Klasse lediglich deklariert aber nicht implementiert.
- g.) Was ist bei Deklaration und Definition von Methoden zu beachten, die den Zustand eines Objektes nicht verändern? **Lösung:**
Sie sind als `const` zu vereinbaren.
- h.) Geben Sie ein (ganz kurzes) Beispiel für ein Funktionsobjekt an? **Lösung:**

```
class A {
    int a;
    public:
    int operator() (float f) {return a*f;}
};
```

Aufgabe 5 : STL

ca. 14 Punkte

- a.) Sie brauchen einen Container. In welchen Fällen ist ein **Vektor** gegenüber einer **Liste** zu bevorzugen?

Lösung:

Ein Vektor ermöglicht den Direktzugriff auf Elementen mit konstantem Zeitaufwand $O(1)$. Muss also häufig auf bestimmte Elemente zugegriffen werden, deren Position bekannt ist bzw. die in einem sortierten Container mit binärer Suche gesucht werden können, ist ein Vektor zu bevorzugen. Wird dagegen häufig ein Element am Anfang oder irgendwo in der Mitte zu entfernen oder neu einzufügen sein, so ist eine Liste der bessere Container gegenüber einem Vektor. In einer Liste ist dieses mit $O(1)$ möglich, im Vektor dagegen nur mit $O(N)$.

- b.) Sie brauchen einen Container. In welchen Fällen ist die Verwendung einer **Map** gegenüber einer **Liste** zu bevorzugen? **Lösung:**

Wenn sowohl häufig nicht gerade am Ende eingefügt wird als auch häufig auf bestimmte Positionen in einem sortierten Container zugegriffen werden soll, so ist eine Map oder ein Set die beste Wahl. Beide ermöglichen sowohl für Einfügen/Löschen als auch den Zugriff jeweils logarithmischen Aufwand, während sowohl Vektor als auch Liste für zumindest eine Operation linearen Aufwand ($O(N)$) erfordern, die andere Operation unterstützen sie aber mit konstantem Aufwand.

- (c.) Was gibt das folgende Programm in die Datei `datei` aus? Sie können die Ausgabe direkt im Programmcode notieren.

Listing 11: (`./Code1/mainMap.cxx`)

```
// Hier stehen die erforderlichen Includes drin
#include "MainMapIncludes.h"

// globale Variable fuer Ausgabedatei kann entsprechend cout verwendet werden
ofstream datei ("map.txt", ios::out);
void vectorFunk();
int main() {
    /***** Teil 1 *****/
    map<int, string> m;
    typedef map<int, string>::const_iterator CIter;
    m[1] = "hallo";
    m[7] = "Meier";
    m[-4] = "Fritz";
    m[4] = "Otto";
    m[0] = "Otto";

    CIter j;
    for (j =m.begin(); j !=m.end(); ++j) {
        datei << j->second << ":" << j->first << endl;
    }

    datei << "#####" << endl;
    for (j =--m.end(); j !=m.begin()++; --j) {
        datei << j->second << ":" << j->first << endl;
    }

    datei << "#####" << endl;
    CIter it = m.find(1);
    for (j=it; j !=m.end(); ++j) {
        datei << j->second << ":" << j->first << endl;
    }
}
```



```

vectorFunk();

return 0;
}

/***** Teil 2 *****/
class FunkObj{
    int value;
public:
    FunkObj(int w): value(w) {}
    int operator() () {return ++value;}
};

void vectorFunk() {
    vector<int> vec;
    generate_n(back_inserter(vec), 4, FunkObj(7));
    vector<int>::iterator vit;
    datei << "\n#####\n";
    for (vit = vec.begin(); vit != vec.end(); ++vit) {
        datei << *vit << ",";
    }

    datei << "\n#####\n";
    copy (vec.begin(), vec.end(),
          ostream_iterator<int>(datei,","));

    vit = vec.end(); vit--;
    generate(vec.begin(), vit, FunkObj(-3));
    datei << "\n#####\n";
    copy (vec.begin(), vec.end(),
          ostream_iterator<int>(datei,","));
}

```

Listing 12: (./Code1/map.txt)

```

Fritz:-4
Otto:0
hallo:1
Otto:4
Meier:7
#####
Meier:7
Otto:4
hallo:1
Otto:0
#####
hallo:1
Otto:4
Meier:7

#####
8,9,10,11,
#####
8,9,10,11,
#####
-2,-1,0,11,

```

Aufgabe 6 : Templates

ca. 13 Punkte

Implementieren Sie eine Klasse `ValueDebug`, die dazu verwendet werden kann, am Ende eines Gültigkeitsbereichs den Wert einer überwachten Variablen mit dem Ausgabeoperator `operator<<` auszugeben. Außerdem soll der Wert der überwachten Variablen beim Aufruf des Konstruktors von `ValueDebug` ausgegeben werden. Die Ausgaben sollen in die globale Datei `datei` oder (wenn Ihnen das lieber ist) auf den Bildschirm (`cout`) erfolgen.

Hinweis: Gehen Sie analog der Implementierung der Klasse `FunktionLog` bzw. `auto_ptr` in der Vorlesung vor (geschickte Nutzung des automatischen Destruktors-Aufrufs am Ende des Gültigkeitsbereichs von Variablen). Verwenden Sie die Klasse `string` zur Textspeicherung.

Das folgende Programm zeigt eine beispielhafte Anwendung der Klasse, die in der Datei `Debug.h` implementiert sein soll.

Listing 13: (`./Code1/mainValueDebug.cxx`)

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

#include "Debug.h"
ofstream datei("ValueDebug.txt", ios::out);

int main()
{
    int i=10;
    double d=17.2;
    ValueDebug<int> iii(i,"i1");
    if (i<13)
    {
        ValueDebug<int> yyy(i, "i2");
        ValueDebug<double> xxx(d, "d");
        d = 22.123;
        i= 144;
    }
    ValueDebug<int> jjj(i, "i3");
    i= 44;

    return 0;
}
```

Dieses Programm soll die folgende Ausgabe erzeugen:

Listing 14: (`./Code1/ValueDebug.txt`)

```
Anfangswert von i1:10
Anfangswert von i2:10
Anfangswert von d:17.2
    Endwert von d:22.123
    Endwert von i2:144
Anfangswert von i3:144
    Endwert von i3:44
    Endwert von i1:44
```

Listing 15: (`./Code1/Debug.h`)

```
extern ofstream datei;

template <class TYP>
class ValueDebug
{
public:
    inline ValueDebug(const TYP& wert, string n);
    inline ~ValueDebug();
private:
    const TYP& var;
    string name;
};

template <class TYP>
ValueDebug<TYP>::ValueDebug
    (const TYP& wert, string n):
    var(wert), name(n)
{
    datei << "Anfangswert_\u von_\u" << name << ":"
        << var << endl;
}

template <class TYP>
ValueDebug<TYP>::~ValueDebug()
{
    datei << "\u\u\u\u Endwert_\u von_\u" << name << ":"
        << var << endl;
}
```