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

Name:						
Matr.-Nr.:		F1		F2		F3	
Ergebnis:						

Klausur im SS 2001 :

Informatik III

Hilfsmittel sind bis auf Computer erlaubt !	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.
Bitten 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 Einfachheithalber die Variable *cout* angenommen werden. Die Variable *datei* diente lediglich bei der Klausurerstellung dem Zweck der Ausgabeumlenkung.

Aufgabe 1 : Wertezuweisung

6 Punkte
Gegeben ist der folgende Ausschnitt aus einem Programm:

```
/*-----------------------------------------------*/
typedef int* PINT;
void func1(double d,  int i);
void func2(PINT pi, int feld[]);
void func3(double** ppd);
void func4(int i, void (*x)(double, double*));
int   func5(int& x, PINT& y);
void*  func6(int&i, int j, int* k);

void sin1(double, double);
void sin2(double);
double sin3(double, double*);
void sin4(double, double*);

int main() {
int i, j, k;
int* pi=new int;
double d1;
double* pd=new double;
float ffeld[]= {1.0f, 2.0f,3.0f};
int ifeld[3000];


func1(d1, i);
func2(&i,  ifeld);
func3(&pd);
func4(i, sin4);
func5(i, pi);
func6(i, j, &k);
}
```

Rufen Sie die Funktionen `func1`, `func2`, `func3`, `func4`, `func5`, `func6` mit korrekten Parametern auf, sodass das C++-Programm syntaktisch korrekt wird und keine Compilerwarnungen erzeugt werden. Verzichten Sie auf jedes explizite *typecast*.
Schreiben Sie den Code der Aufrufe direkt im Programm an der dort freigehaltenen Stelle.

Aufgabe 2 : Kontrollstrukturen

2 + 2 Punkte
Was gibt das folgende Programm in die Datei *datei* aus?

```
#include <fstream>
using namespace std;
ofstream datei("test.txt", ios::out);

void func1(int i)
{
for (int j=i; j<6; j=j+2)
   {
   datei << j << " ";
   }

for (; i<4; ++i)
   {
   datei << i << " ";
   }

}

int main()
{
func1(0);
datei << endl;
func1(4);
datei << endl;
return 0;
}
```

```
Ausgabe:
0 2 4 0 1 2 3
4
```

Aufgabe 3 : Funktionen

5 Punkte
Implementieren Sie eine Funktion *Sum*, der eine beliebige Anzahl ganzer Zahlen (`int`) und deren Anzahl übergeben wird. Der Rückgabewert dieser Funktion ist die Summe

der übergebenen ganzen Zahlen. In ihrem dritten Parameter gibt die Funktion die Summe aller übergebenen positiven Zahlen zurück. Garantieren Sie dem Benutzer der Funktion das keiner der beiden ersten Eingangsparameter in der Funktion geändert wird.

```
#include <fstream>
using namespace std;
ofstream datei("test.txt", ios::out);



int  Sum(const int feld[], int anzahl, int& posSum);

int main()
{
int feld[]={1,2,3,-3,-2,-4};
int poSum;
int geSum=Sum(feld,6,poSum);
datei << "Gesamtsumme: " << geSum << endl;
datei << "Positive Summe: " << poSum << endl;

return 0;
}

/*
Das Programm soll folgende Ausgabe liefern:
Gesamtsumme: -3
Positive Summe: 6
*/
```

Es gibt einen Zusatzpunkt, wenn der Parameter `feld` als `const int` übergeben wird.

```
int  Sum(const int feld[], int anzahl, int& posSum)
{
posSum = 0;
int geSum = 0;
for (int i = 0; i < anzahl; ++i)
   {
  geSum += feld[i];
  if (feld[i]>0)
     {
     posSum += feld[i];
     }
   }
return geSum;
}
```

Aufgabe 4 : Objektorientierte Programmierung

(3+2+3+3+4+8+3) Punkte
Gegeben sind die folgenden Klassendefinitionen:

```
class Tier
   {
   int alter;
   public:
   Tier(int a=0): alter(a) {
       datei << "+T" << alter << " ";}
   virtual ~Tier() {datei << "-T" << alter << " ";}
   virtual void Typ() const {datei  << " T ";}
   void FussAnzahl() const {datei  << " 4 ";}
   };

class Vogel: public Tier
   {
   int groesse;
   public:
   Vogel(int gr, int a):  Tier(a),  groesse(gr)
      {datei << "+V" << groesse << " ";}
   Vogel(int gr): groesse(gr)
      {datei << "+V" << groesse << " ";}
   virtual ~Vogel() {datei << "-V" << groesse << " ";}
   virtual void Typ() const {datei  << " V ";}
   void FussAnzahl() const {datei  << " 2 ";}
   };
```

a.) Was gibt das folgende Programmfragment in die Datei *datei* aus?

```
void func1()
{
Tier tier(1);
Vogel vogel(2,7);
}

int main(){func1();}
```

```
+T1 +T7 +V2 -V2 -T7 -T1
```

b.) Was gibt das folgende Programmfragment in die Datei *datei* aus?

```
void func2()
{
Tier tier;
Vogel vogel(3);
}

int main(){func2();}
```

```
+T0 +T0 +V3 -V3 -T0 -T0
```

c.) Was gibt das folgende Programmfragment in die Datei *datei* aus?

```
void Print(Tier t)
{
t.Typ();  t.FussAnzahl();
}

void func3()
{
Tier t1(5);
Print(t1);
}

int main(){func3();}
```

```
+T5  T  4 -T5 -T5
```

d.) Was gibt das folgende Programmfragment in die Datei *datei* aus?

```
void func4()
{
Tier* pt = new Vogel(5,5);
pt-> Typ();   pt->FussAnzahl();
delete pt;
}

int main(){func4();}
```

```
+T5 +V5  V  4 -V5 -T5
```

e.) Was gibt das folgende Programmfragment in die Datei *datei* aus?

```
void func5()
{
Tier tfeld[3];
}
int main(){func5();}

+T0 +T0 +T0 -T0 -T0 -T0
```

f.) Was gibt das folgende Programmfragment in die Datei *datei* aus?

```
void func6()
{
Vogel   *pvfeld[3];
pvfeld[1] = new Vogel(4,4);
pvfeld[1]-> Typ(); pvfeld[1]-> FussAnzahl();
datei << endl;

Tier t (*pvfeld[1]);
t. Typ(); t. FussAnzahl();
Tier* pt = pvfeld[1];
pt-> Typ(); pt-> FussAnzahl();
delete pt;
}

int main(){func6();}

+T4 +V4  V  2
 T  4  V  4 -V4 -T4 -T4
```

g.) Warum ist das folgende Programmfragment syntaktisch falsch? Ändern/Erweitern Sie die Klasse *Tier* bzw. *Vogel*, sodass es zumindest syntaktisch richtig wird.

```
void funcFalsch()
{
Vogel vfeld[3];
}
```

Die Klasse *Vogel* hat keinen Standard-Konstruktor. Erweiterung der Klasse Vogel um einen Standard-Konstruktor.

```
Vogel() {};
```

Aufgabe 5 : Operatoren und Copy-Konstruktor

3 + 2 + 3 Punkte

a.) Schreiben Sie für die Klasse *Tier* der vorherigen Aufgabe einen Zuweisungsoperator. Vermeiden Sie unnötige Operationen.

b.) Schreiben Sie für die Klasse *Tier* der vorherigen Aufgabe einen Vergleichsoperator als Elementfunktion (2 Tiere sind gleich, wenn ihr Alter gleich ist).

c.) Schreiben Sie für die Klasse *Tier* der vorherigen Aufgabe einen Copy-Konstruktor.

Hinweis: Sie können alle Methoden dieser drei Teilaufgaben `inline` implementieren, d.h. es wird angenommen, dass der von Ihnen hier implementierte Code im public-Teil nach der Methode `Tier::FussAnzahl` folgt.
Das folgende Programm soll dann übersetzbar sein

```
Tier t1(5);
Tier t2(6);
Tier t3(t1);
if (t1 == t3) {datei << "t1 == t3 "; }
    else  {datei << "t1 != t3 "; }
if (t1 == t2) {datei << "t1 == t2 "; }
    else  {datei << "t1 != t2 "; }
t3=t2;
t1=t2=t3;
if (t1 == t2) {datei << "t1 == t2 "; }
    else  {datei << "t1 != t2 "; }
```

und die folgende Ausgabe erzeugen, wenn die Ausgaben der Konstruktoren und Destruktoren ignoriert werden.

```
t1 == t3 t1 != t2 t1 == t2
```

```
   Tier& operator=(const Tier& t2)
      {alter = t2. alter; return *this;}
   Tier(const Tier& t2): alter(t2.alter) {;}
   bool operator==(const Tier& t2)
      {return (alter == t2. alter? true : false);}
```

Aufgabe 6 : Header- und Quellcode-Dateien, Templates

11 Punkte

Implementieren Sie eine Klasse `ObjBenutzung`, die ein Objekt (z.B. eine Instanz einer Klasse) für eine exklusive Benutzung sperrt (belegt), sodass z.B. andere Prozesse nicht mehr *gleichzeitig* auf dieses Objekt zugreifen können. Nach Benutzung des Objektes muss das Objekt selbstverständlich wieder freigegeben werden.
Damit dieses nicht vergessen wird und die Benutzung einfach und komfortabel ist, soll das Objekt *automatisch* durch die Klasse `ObjBenutzung` wieder freigegeben werden, sobald der Gültigkeitsbereich einer Instanz von `ObjBenutzung` verlassen wird, d.h. spätestens bei Beendigung des benutzenden Programms.
Hinweis: Gehen Sie analog der Implementierung der Klasse `FunktionLog` in der Vorlesung vor.
Implementieren Sie die Deklaration der Klasse in der Header-Datei *ObjBenutzung.h* und deren Definition in der Datei *ObjBenutzung.cpp*. **Vergessen Sie nicht** die notwendigen `include`-Anweisungen und einen Include-Wächter zu implementieren. Vermeiden Sie aber unnötige `include`-Anweisungen.
Beispielanwendung

```
/* Einbinden der Source-Datei, damit
   es keine Probleme bei Erzeugen der
   Template-Instanzen von ObjBenutzung
   gibt. */
#include "ObjBenutzung.cpp"

class Somewhat{
int mi;
int mfeld[199];
};

Somewhat g_some1;
int g_i= 7;
Somewhat g_some2;

int main()
{
ObjBenutzung<Somewhat> ben1(g_some1);
if (g_i < 10)
   {
   ObjBenutzung<int> ben2(g_i);
```

```
   /* ....*/
   }
ObjBenutzung<Somewhat> ben3(g_some2);
/* :::*/

return 0;
}
```

Gehen Sie davon aus, dass die (auch in der Vorlesung vorgestellten) Schablonenfunktionen `Belege` und `Freigeben` bereits in der Datei *BelegeFrei.h* vollständig als inline-Funktionen implementiert sind, d.h. sie können von der Klasse `ObjBenutzung` verwendet werden.
Ein Ausschnitt aus ihrer Implementierung könnte wie folgt aussehen:

```
template <class TYP>
void Belege(const TYP& obj)
{
/*....   */
datei << "Obj mit " << sizeof(TYP)
         << " Bytes ab Adr " << &obj
          << " wurde belegt." << endl;
};

template <class TYP>
void Freigeben(const TYP& obj)
{
/*....   */
datei << "Obj mit " << sizeof(TYP)
         << " Bytes ab Adr " << &obj
          << " wurde freigegeben." << endl;
};
```

Diese Implementierung soll zusammen mit obigem Hauptprogramm und Ihrer Implementierung von der Klasse `ObjBenutzung` die folgende Ausgabe erzeugen:

```
Obj mit 800 Bytes ab Adr 00453D60 wurde belegt.
Obj mit 4 Bytes ab Adr 0044EDF0 wurde belegt.
Obj mit 4 Bytes ab Adr 0044EDF0 wurde freigegeben.
Obj mit 800 Bytes ab Adr 00454080 wurde belegt.
Obj mit 800 Bytes ab Adr 00454080 wurde freigegeben.
Obj mit 800 Bytes ab Adr 00453D60 wurde freigegeben.
```

Realisierung durch Konstruktor und Destruktor 2 Punkte
Include-Wächter 1 Punkte, Klassen-Schnittstelle 2,5 Punkte
Beide Include-Anweisungen 1,5 Punkte, template class vor Methoden 1 Punkte
m_obj belegen 1 Punkte, Aufruf von Belegen und Freigeben 2 Punkte ObjBenutzung.h

```
#ifndef OBJ_BENUTZUNG_HEADER
#define OBJ_BENUTZUNG_HEADER

template <class TYP>
class ObjBenutzung
{
public:
ObjBenutzung(const TYP& obj);
~ObjBenutzung();
private:
const TYP& m_obj;
};

#endif
```

ObjBenutzung.cpp

```
#include "ObjBenutzung.h"
#include "BelegeFrei.h"

template <class TYP>
ObjBenutzung<TYP>::ObjBenutzung
     (const TYP& obj): m_obj(obj)
{
Belege(m_obj);
}

template <class TYP>
ObjBenutzung<TYP>::~ObjBenutzung()
{
Freigeben(m_obj);
}
```

Aufgabe 7 : STL, Funktionsobjekte

15 Punkte

a.) Was gibt das folgende Programm an den Stellen, die mit /*1*/ /*2*/ /*3*/ /*4*/ /*5*/ /*6*/ gekennzeichnet sind, in die Datei `datei` aus? Die Klasse `pair` ist im Anhang nochmals aufgeführt.

b.) Erweitern Sie das Programm an der Stelle /* Entfernen von 2 */, sodass die 2 aus der Liste entfernt wird, d.h. anschließend nur noch 4 Elemente in der Liste vorhanden sind.

```
#include <fstream>
#include <list>
#include <set>
#include <vector>
#include <string>
#include <utility>  // wegen pair
#include <algorithm>
using namespace std;

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

typedef pair<string, int> T_NameAlter;

class Print
{
public:
Print(int wert=0): m_i(wert) {;}
void operator()(const T_NameAlter& paar)
{
datei << "(" << paar.first << ","
         << paar.second + m_i << ") ";
}
private:
int m_i;
};

int main()
{
list<int>  cont;
for (int i=1; i<6; ++i)
   {
   cont.push_back(i);
   }
/* 1 */
copy(cont.begin(), cont.end(),
  ostream_iterator<int>(datei, " "));
datei << endl;

/* Entfernen von 2 */
```

```
set<T_NameAlter> na_set;
na_set. insert(T_NameAlter("Meier", 18));
na_set. insert(T_NameAlter("Meier", 17));
na_set. insert(T_NameAlter("Meier", 18));
na_set. insert(T_NameAlter("Mueller", 101));

/*2*/
for_each(na_set.begin(), na_set.end(), Print());
datei << endl;

vector<T_NameAlter> na_vec;
Print pri(3);
na_vec. push_back(T_NameAlter("Meier", 18));
na_vec. push_back(T_NameAlter("Meier", 17));
na_vec. push_back(T_NameAlter("Meier", 18));

/*3*/
for_each(na_vec.begin(), na_vec.end(), pri);
datei << endl;

na_vec[1] = T_NameAlter("Mueller", 101);
/*4*/
for_each(na_vec.begin(), na_vec.end(), pri);
datei << endl;

vector<T_NameAlter>::iterator iter =
    find(na_vec.begin(), na_vec.end(),
         T_NameAlter("Mueller", 101));
/*5*/
for_each(na_vec.begin(), iter, pri);
datei << endl;
/*6*/
for_each(iter, na_vec.end(), pri);
datei << endl;

return 0;
}
```

1 Punkt, 3 Punkte, 3 Punkte, 1 Punkt, 1,5 Punkt, 1,5 Punkt, 4 Punkte (erase)

```
1 2 3 4 5
(Meier,17) (Meier,18) (Mueller,101)
(Meier,21) (Meier,20) (Meier,21)
(Meier,21) (Mueller,104) (Meier,21)
(Meier,21)
(Mueller,104) (Meier,21)

cont.erase(remove(cont.begin(), cont.end(), 2),
                  cont.end());
```

Anhang 1: Klasse pair der STL

```
template <class _T1, class _T2>
struct pair {
  typedef _T1 first_type;
  typedef _T2 second_type;

  _T1 first;
  _T2 second;

  pair() : first(_T1()), second(_T2()) {}
  pair(const _T1& __a, const _T2& __b) :
                   first(__a), second(__b) {}
/* ...*/
};
```

```
// Test auf Gleichheit
template <class _T1, class _T2>
inline bool operator==
    (const pair<_T1, _T2>& __x,
     const pair<_T1, _T2>& __y)
   {return __x.first == __y.first &&
           __x.second == __y.second; }

// Test auf kleiner als
template <class _T1, class _T2>
inline bool operator<
     (const pair<_T1, _T2>& __x,
      const pair<_T1, _T2>& __y)
   {return __x.first < __y.first ||
             (!(__y.first < __x.first) &&
               (__x.second <__y.second)); }
```

Geplante Punktevergabe

A1: 6 P	A2: 2+2 P	A3: 5 P
A4: 3+2+3+3+4+8+3=26 P		
A5: 3+2+3 P	A6: 11 P	A7: 15 P
Summe 75 P		