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

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

Probeklausur 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 hocheffiziente 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:

```
/*-----------------------------------------------*/
double sin1(double, double);
double sin2(double);
double sin3(double, double*);
double sin4[22];

int main() {
int i;
int feld[10];
void* pv;
int* pi;
double (*f)(double, double*);
void** ppv;
/* ..*/

i = 4;
feld[4] = 4;
pv = &i;
pi = &i;
f = sin3;
ppv = &pv;
```

Weisen Sie den Variablen `i, feld[4], pv, pi, f, ppv` einen Wert zu, sodass das C++-Programm syntaktisch korrekt wird.
Nehmen Sie die Zuweisungen direkt im Code vor.

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)
{
int help = i;
while (help > 0)
   {
   datei << help << " ";
   help--;
   }
do
   {
   datei << i << " ";
   } while (--i  > 0);
}

int main()
{
func1(0);
datei << endl;
func1(2);
datei << endl;
return 0;
}
```

```
Ausgabe:
0
2 1 2 1
```

Aufgabe 3 : Funktionen

5 Punkte
Implementieren Sie eine Funktion *IstGerade*, die testet, ob eine ganze Zahl (... -3, -2, -1, 0, 1, 2, 3 ...) eine gerade Zahl (0, 2, 4, ...) ist. Wird der Funktion eine negative Zahl übergeben, wird die Funktion *false* als Funktionswert liefern, andernfalls *true*. Falls im letzten Fall der Funktion eine gerade Zahl übergeben wird, soll die Funktion in ihrem zweiten Parameter den Wert *true* zurückliefern, andernfalls (ungerade Zahl) *false*.
Die Funktion soll z.B. im folgenden Programm benutzt werden können.

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

```
bool IstGerade(int zahl, bool& erg);

int main()
{
bool gerade;
bool fwert;

fwert = IstGerade(3, gerade);
...
return 0;
}


// Loesung 1
bool IstGerade(int zahl, bool& erg)
{
erg = (zahl%2) == 0? true : false;
return ((zahl>=0)? true : false);
}

// Alternativloesung
bool IstGerade(int zahl, bool& erg)
{
if (zahl < 0)
   {
   return false;
   }
else
   {
   if ((zahl % 2) == 0)
      {
      erg = true;
      }
  else
      {
      erg = false;
      }
   } // else von if (zahl < 0)
return true;
}
```

Aufgabe 4 : Operatoren

2 + 7 + 2 + 2 Punkte

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

class Stack
{
public:
   Stack(int size): max(size), n(0)
       { data = new int [max]; }
   ~Stack() {delete[] data; }
   void Push(int zahl) {data[n++]=zahl;   }
   int Pop() {return data[--n] ;}
   int RestPlaetze()  {return (max - n);}
private:
   int* data;
   int max;
   int n; // aktuelle Elementanzahl

   friend inline ostream& operator<<
         (ostream& str, const Stack& st);
};

/* Hier fehlt der Ausgabeoperator */

int main()
{
Stack st1(5);
Stack st2(7);
st1. Push(1);
st2. Push(2);
st1. Push(3);
datei << st1 << endl << st2 << endl;
datei << "Pop von 3: " << st1.Pop() << endl;
st1.Pop();
datei << st1 << endl << st2 << endl;
return 0;
}
```

Obiges Programm soll die folgende Ausgabe produzieren:

```
[1 3 ] Noch 3 Plaetze frei.
[2 ] Noch 6 Plaetze frei.
Pop von 3: 3
[] Noch 5 Plaetze frei.
[2 ] Noch 6 Plaetze frei.
```

a.) Welche Methoden der Klasse `Stack` könnten als konstante Elementfunktionen vereinbart werden? `RestPlaetze` ist die einzige konstante Methode.

b.) Implementieren Sie den Ausgabeoperator `operator<<` der Klasse `Stack`, wobei Sie davon ausgehen können, dass alle Methoden, die als konstant implementiert werden können, auch konstant implementiert sind, d.h. die Sie im Teil a.) als konstant implementieren wollen.

```
inline ostream& operator<<
   (ostream& str, const Stack& st)
{
str << "[";
for (int i=0; i < st.n; ++i)
   {
   str << st.data[i] << " ";
   }
str << "]";
str << " Noch " << st.RestPlaetze()
      << " Plaetze frei.";
return str;
}
```

Damit `st` als konstanter Parameter übergeben werden kann, muss die Methode `Stack::RestPlaetze` eine konstante Methode sein.

c.) Warum kann die Methode *Stack::Push* **keine** konstante Elementfunktion sein? Durch `Push` wird das Element `Stack::n` verändert. Deshalb ist es keine konstante Methode. Anmerkung:
Eine Veränderung von `Stack::data` findet durch `Push` nicht statt. Es wird lediglich der Speicherbereich, auf den `Stack::data` verweist, durch `Push` verändert. Diese würde somit nicht geben eine konstante Methode `Push` sprechen.

d.) Warum ist es wichtig, dass in dem Programm in der vorliegenden Form im Operator `operator<<` der Parameter vom Typ *Stack* als Referenzparameter übergeben wird? Andernfalls würde der Copy-Konstruktor benötigt. Dieser ist nicht implementiert. Es würde ein Default-Copy-Konstruktor vom System generiert, was in diesem Fall zu undefiniertem Verhalten führen würde, da zweimal `delete` für die gleiche Speicheradresse aufgerufen würde.

Aufgabe 5 : Polymorphie

(2+2+2+2+2) + 2 Punkte

a.) Was gibt das folgende Programm aus?

```
T  4
V  2
T  4  V  2
T  4  T  4
T  4  T  4
```

b.) Was ändert sich an der Ausgabe des Programms, wenn man die Parameter an die drei Funktionen `PrintTierRef`, `PrintTierWert` und `PrintTierWertRef` jeweils als `const`-Parameter übergibt und sonst alles so lässt, d.h.:

```
void PrintTierRef(const Tier& rt)   /* ...*/
void PrintTierWert(const Tier t)    /* ...*/
void PrintTierWertRef(const Tier t) /* ...*/
```

nichts

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

class Tier
   {
   public:
   virtual void Typ() const {datei  << " T ";}
   virtual void FussAnzahl() const {datei  << " 4 ";}
   };

class Vogel: public Tier
   {
   public:
   virtual void Typ() const {datei  << " V ";}
   virtual void FussAnzahl() const {datei  << " 2 ";}
   };

void PrintTierRef(Tier& rt)
{
rt. Typ();     rt. FussAnzahl();
}

void PrintTierWert(Tier t)
{
t. Typ();     t. FussAnzahl();
}

void PrintTierWertRef(Tier tier)
{
PrintTierRef(tier);
}


int main()
{
Tier sam;
Vogel tweety;

sam. Typ(); sam. FussAnzahl(); datei << endl;
tweety. Typ(); tweety. FussAnzahl(); datei << endl;

PrintTierRef(sam); PrintTierRef(tweety); datei<<endl;
PrintTierWert(sam); PrintTierWert(tweety); datei<<endl;
PrintTierWertRef(sam); PrintTierWertRef(tweety);
   datei<<endl;
return 0;
}
```

Aufgabe 6 : Vererbung

7 Punkte

Leiten Sie von der Klasse `Vogel` der Aufgabe eine Klasse `Strauss` ab. Die Klasse `Strauss` soll zusätzlich zu den Methoden von `Vogel` eine öffentliche Methode `LaufVogel` haben, die immer `true` liefert. Implementieren Sie eine vollständige Header-Datei (mit Include-Wächter) und Quellcode-Datei (Cpp-Datei). Die Methode von `Strauss` soll in der Quellcodedatei implementiert werden. Die Klasse `Vogel` soll in der Header-Datei `Vogel.h` deklariert sein.

Hinweis: Denken Sie an die Benutzung des Schlüsselwortes `const`.

```
// Datei Strauss.h

#include "Vogel.h"
#ifndef STRAUSS_HEADER
#define STRAUSS_HEADER
class Strauss: public Vogel
{
public:
bool Laufvogel() const;
};

#endif

// Datei Strauss.cpp
#include "Strauss.h"

bool Strauss::Laufvogel() const
{
return true;
}
```

Aufgabe 7 : Header- und Quellcode-Dateien

4 + 2 Punkte

a.) Gegeben ist folgende Implementierung einer Klasse `Stack`. Diese ist in der Header-Datei `stack.h` und einer Quellcode-Datei `stack.cpp` implementiert. Leider sind beim Kopieren die beiden Dateien *durchmischt* worden. Geben Sie daher hinter jeder Zeile an, ob diese in die Header-Datei oder in die Quellcode-Datei gehören.

/*_____*/

Notieren Sie die Zeilennummern der Header- und Quellcode-Datei hier.

/*_____*/

b.) Ergänzen Sie die Header-Datei oder die Quellcode-Datei um einen Include-Wächter. Header-Datei: Zeilen 1-14, 17-18

Source-Datei:
Zeilen 15-16, 19-23

Include-Wächter: Vor Zeile 1:

```
#ifndef STACK_HEADER
#define STACK_HEADER
```

Zwischen Zeile 18 und 19:

```
#endif
```

```
class Stack {                              // 1
  public:                                  // 2
    Stack  (int siz=DEFAULT_INIT_SIZE);    // 3
    ~Stack ();                             // 4
    void Push (int x);                     // 5
    inline int  Pop  ();                   // 6
    inline int count () const;             // 7
  protected:                               // 8
    int *Data;                             // 9
    int n;                                 // 10
    int size;                              // 11
    private:                               // 12
    static int DEFAULT_INIT_SIZE;          // 13
};                                         // 14


#include "stack3.h"                        // 15
int Stack::DEFAULT_INIT_SIZE=10;           // 16

inline int  Stack::Pop   ()       {
   return Data[--n]; }                     // 17
inline int  Stack::count () const {
   return n; }                             // 18



Stack::Stack  (int siz)  {                 // 19
 Data = new int[size=siz]; n = 0; }        // 20
Stack::~Stack ()  { delete[] Data; }       // 21

void Stack::Push  (int x)  {               // 22
    Data[n++] = x; }                       // 23
```

Aufgabe 8 : Templates

10 Punkte

Schreiben Sie eine Klasse `ValueDebug`, die dazu verwendet werden kann, am Ende eines Gültigkeitsbereichs den Wert einer überwachten Variablen auszugeben, sofern für diese Variable ein Ausgabeoperator `operator<<` definiert ist. Außerdem soll der Wert der überwachten Variablen beim Aufruf des Konstruktors von `ValueDebug` ausgegeben werden. Die Ausgabe soll 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` in der Vorlesung vor. Verwenden Sie die Klasse `string`.

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

```
#include "ValueDebug.h"

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:

```
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
```

```
// Datei ValueDebug.h
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 von " << name << ":"
         << var << endl;
}

template <class TYP>
ValueDebug<TYP>::~ValueDebug()
{
datei << "    Endwert von " << name << ":"
         << var << endl;
}
```

Aufgabe 9 : STL, Container

5 Punkte

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

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

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

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

remove(cont.begin(), cont.end(), 3);
datei << "Liste nach remove :";
copy(cont.begin(), cont.end(),
     ostream_iterator<int>(datei, " "));
datei << endl;
return 0;
}
```

```
Liste vor remove   :1 2 3 4 5
Liste nach remove :1 2 4 5 5
```

Aufgabe 10 : STL, Funktionsobjekte

7 Punkte
Was gibt das folgende Programm in die Datei `datei` aus?

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

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

class FibboFolge
{
private:
    int mi_vorlWert;
    int mi_letzWert;
public:
FibboFolge (int ai_startwert); // Konstruktor
    int operator() ();
};

FibboFolge::FibboFolge (int ai_startwert)
{
mi_vorlWert = 0;
mi_letzWert = 1;
while ((ai_startwert--) > 0)
   {   operator()();   }
}

int FibboFolge::operator() ()
{
int temp  = mi_letzWert;
mi_letzWert += mi_vorlWert;
mi_vorlWert = temp;
return temp;
}

int main()
{
list<int> menge;
FibboFolge fib(3);
for (int i=0; i<7; ++i)
   {
   int elem = fib();
   datei << elem << " ";
   back_inserter(menge)=elem;
   // entspricht menge.push_back(elem);
   }
datei << endl;

copy (menge.begin(),   menge.end(),
        ostream_iterator<int>(datei," "));
datei << endl;

generate_n (back_inserter(menge),    // Anfang
                  4,                 // Anzahl
                  FibboFolge(1));
copy (menge.begin(),   menge.end(),
        ostream_iterator<int>(datei," "));
datei << endl;

return 0;
}
```

```
3 5 8 13 21 34 55
3 5 8 13 21 34 55
3 5 8 13 21 34 55 1 2 3 5
```

Geplante Punktevergabe

A1: 6 P	A2: 2+2 P	A3: 5 P	A4:
A5: 10+2 P	A6: 7 P	A7: 4+2 P	A8:
A9: 5 P	A10: 7		
Summe 75 P			