# Name:



Hon. Prof. Dr.-Ing. Hartmut Helmke
Ostfalia
Hochschule für angewandte Wissenschaften
Fakultät für Informatik

*Matrikelnummer:* *Punktzahl:*

*Ergebnis:*

*Freiversuch* ☐ *F1* ☐ *F2* ☐ *F3* ☐

Klausur im WS 2019/20:

## Weitere Programmiersprache — Lösungen —

*Informatik Bachelorstudiengang* ☐ *Informatik Masterstudiengang* ☐

| | |
|---|---|
| Hilfsmittel sind bis auf Computer, Handy etc. erlaubt ! | Bitte Aufgabenblätter mit abgeben ! |
| Austausch von Hilfsmitteln mit Kommilitonen ist **nicht** erlaubt ! | |
| Anwesenheit von Handys, Smartphones etc. bei Klausurteilnehmern im Hörsaal nicht erlaubt. | |
| Sie sind vor Beginn der Klausur am Dozentenpult abzugeben! | |

Bitte notieren Sie auf **allen** 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 Programmfragmenten wird manchmal die globale Variable *datei* verwendet. Hierfür kann der Einfachheit halber die Variable *cout* angenommen werden. Die Variable diente bei der Klausurerstellung lediglich dazu, automatisch eine Lösungsdatei zu erstellen. Wir befinden uns jeweils im Namensraum `std`, d.h. ein `using namespace std;` dürfen Sie in jeder Codedatei annehmen.

Alle Lösungen – außer für Aufgabe 1 – sind auf Extrablättern mit vorangestellter Aufgabennummer zu notieren. Bitte vergessen Sie nicht, Ihren Namen auf den Extrablättern zu notieren.

## Geplante Punktevergabe

| Punktziel | Im einzelnen | Pkte |
|---|---|---|
| Tests/Team: | xxx | |
| A1: 2 P. | xxx | |
| A2: 28 P. | | |
| A3: 42 P. | | |
| A4: 8 P. | | |
| Summe 80+ SP. | | |

### Aufgabe 1 : Geschenk

ca. 2 Punkte Notieren Sie jetzt auf Seite 1 Ihre Matrikelnummer und Ihren vollständigen Namen in die dafür vorgesehenen Felder. Notieren Sie auch auf allen Aufgabenblättern und zusätzlichen Blättern zumindest Ihren vollständigen Namen.

### Aufgabe 2 : STL etc.

ca. 28 Punkte
Achtung: Im Unterschied zu vorherigen Klausuren sind die Ausgaben auf mehrere Teilfragen aufgeteilt.
Zur Ausgabe von `int` bzw. Instanzen von `string` verwenden wir im Folgenden die beiden Funktionen:

```
// Ausgabe von w plus Komma in Stream datei
void Print(int w) {
   datei << w << ", ";
}
void PriStr(string w) {
   datei << w << ", ";
}
```

**a.) (2 P.)** Zu welcher Ausgabe nach `datei` führt der Aufruf der folgenden Funktion `STL1` nach Ausführung von `for_each` an der Stelle / * 1 */?
**Lösung:**

```
3, 5, 6, 6,
```

`remove` ist ein Algorithmus, der nicht `begin` und `end` verändern kann. Er kann nur die Inhalte der Elemente im Container ändern.
**b.) (2 P.)** Welche Ausgabe ergibt die Ausführung von `for_each` an der Stelle / * 2 */?
**Lösung:**

```
6,
```

```
void STL1() {
   vector<int> v;
   for (int j = 3; j < 7; ++j) {
      v.push_back(j);
   }
   // Ausgabe von: 3, 4, 5, 6,
   for_each(v.begin(), v.end(), Print);
   datei << endl;
   auto newEnd = remove(v.begin(), v.end(), 4);
   for_each(v.begin(), v.end(), Print); /* 1 */
   datei << endl;
   for_each(newEnd, v.end(), Print); /* 2 */
   datei << endl;
}
```

**c.) (1 P.)** Zu welcher Ausgabe nach `datei` führt der Aufruf der folgenden Funktion `STL2` nach Ausführung von `for_each` an der Stelle / * 1 */?
**Lösung:**

```
1, 1, 2, 1, 2, 2, 3,
```

`unique` ist ein Algorithmus, der nicht `begin` und `end` verändern kann. Er kann nur die Inhalte der Elemente im Container ändern. `unique` hat auch nur Auswirkungen für aufeinander folgende identische Elemente.

**d.) (2 P.)** Welche Ausgabe ergibt die Ausführung von `for_each` an der Stelle / * 2 */?

**Lösung:**

```
1, 2, 1, 2, 3,
```

**e.) (2 P.)** Welche Ausgabe ergibt die Ausführung von `for_each` an der Stelle / * 4 */?

**Lösung:**

```
1, 1, 2, 2, 2, 3, 3,
```

**f.) (2 P.)** Warum ist der auskommentierte Aufruf von `sort` an der Stelle 3a ein Syntaxfehler und an der Stelle 3b ist es korrekt? Berücksichtigen Sie bei Ihrer Antwort, dass für eine Instanz von `vector` die Version 3a korrekt wäre und auch zum gewünschten Ergebnis führt.

**Lösung:**
Der generische Algorithmus `sort` implementiert den Quicksort-Algorithmus. Dieser verlangt Direkt-Zugriffs-Iteratoren, d.h. es muss möglich sein, auf das j-te Element im Container effizient (d.h. O(1)) zuzugreifen. Das ist für Instanzen von `list` nicht möglich. Deshalb gibt es für Listen eine spezielle Implementierung der Elementfunktion `sort`.

```
void STL2() {
   list<int> li = { 1, 1, 2, 1, 2, 2, 3 };
   for_each(li.begin(), li.end(), Print); /* 1 */
   datei << endl;
   auto newEnd = unique(li.begin(), li.end());
   for_each(li.begin(), newEnd, Print); /* 2 */
   datei << endl;
   // sort(li.begin(), li.end()); /* 3a */
   li.sort(); /* 3b */
   for_each(li.begin(), li.end(), Print); /* 4 */
   datei << endl;
}
```

**g.) (2 P.)** Zu welcher Ausgabe nach `datei` führt der Aufruf der folgenden Funktion `STL3` nach Ausführung von `for_each` an der Stelle / * 1 */?

**Lösung:**

```
1, 2, 2, 2, 3, 3, 4,
```

**h.) (2 P.)** Welche Ausgabe ergibt die Ausführung von `for_each` an der Stelle / * 2 */?

**Lösung:**

```
1, 2, 3, 4,
```

```
void STL3() {
   array<int, 7> arr = { 1, 2, 2, 3, 4, 2, 3 };
   sort(arr.begin(), arr.end());
   for_each(arr.begin(), arr.end(), Print); /* 1 */
   datei << endl;
   auto newEnd = unique(arr.begin(), arr.end());
   for_each(arr.begin(), newEnd, Print); /* 2 */
   datei << endl;
}
```

**i.) (2 P.)** Zu welcher Ausgabe nach `datei` führt der Aufruf der folgenden Funktion `STL4` nach Ausführung von `for_each` an der Stelle / * 1 */?
**Lösung:**

```
11, 12, 2, 9,
```

**j.) (2 P.)** Welche Ausgabe ergibt die Ausführung von `for_each` an der Stelle / * 2 */?
**Lösung:**

```
11, 12, 2, 9,
```

**k.) (2 P.)** Welche Ausgabe ergibt die Ausführung von `for_each` an der Stelle / * 3 */?
**Lösung:**

```
19, 20, 10, 17,
```

```
void STL4() {
   vector<int> vec = { 11, 12, 2, 9 };
   for_each(vec.begin(), vec.end(), Print); /* 1 */
   datei << endl;
   for (auto iter : vec) {
      iter = iter + 6;
   }
   for_each(vec.begin(), vec.end(), Print); /* 2 */
   datei << endl;
   for (auto& iter : vec) {
      iter = iter + 8;
   }
   for_each(vec.begin(), vec.end(), Print); /* 3 */
   datei << endl;
}
```

**l.) (1 P.)** Zu welcher Ausgabe nach `datei` führt der Aufruf der folgenden Funktion `STL5` nach Ausführung von `for_each` an der Stelle / * 1 */?
**Lösung:**

```
Xavier, Otto, Berta, Ann,
```

**m.) (3 P.)** Welche Ausgabe ergibt die Ausführung von `for_each` an der Stelle / * 2 */?
**Lösung:**

```
Berta, Otto, Xavier, Ann,
```

**n.) (3 P.)** Welche Ausgabe ergibt die Ausführung von `for_each` an der Stelle / * 3 */?
**Lösung:**

```
Ann, Otto, Berta, Xavier,
```

```
void STL5() {
   vector<string> vec =
      { "Xavier", "Otto", "Berta", "Ann" };
   for_each(vec.begin(), vec.end(), PriStr); /* 1 */
   datei << endl;
   sort(vec.begin()++, --vec.end());
   for_each(vec.begin(), vec.end(), PriStr); /* 2 */
   datei << endl;

   sort(vec.begin(), vec.end(),
      [](string s1, string s2) {
         return s1.length() < s2.length(); }
   );
   for_each(vec.begin(), vec.end(), PriStr); /* 3 */
   datei << endl;
}
```

## Aufgabe 3 : Instanzerzeugung

ca. 42 Punkte

Für die folgenden Aufgabenteile ist folgende Klassendeklaration gegeben:

```
class Person {
public:
   Person(string s) {na = s; datei<< " +P "<< na;}
   Person() {na = "?"; datei << " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop " << na;}
   ~Person() { datei << " -P " << na; }
private:
   string na;
};
```

**a.) (3 P.)** Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `Erz1` ?

```
void Erz1() {
   Person ha("Hans");
   Person ka("Karl");
}
```

**Lösung:**

```
+P Hans +P Karl -P Karl -P Hans
```

**b.) (3 P.)** Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `Erz2` ?

```
void Erz2() {
   Person ha("Hans");
   Person copy(ha);
}
```

**Lösung:**

```
+P Hans +PCop Peter -P Peter -P Hans
```

**c.) (2 P.)** Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `Erz3` ?

```
void Erz3() {
   Person* pha = new Person("Hans");
   Person* p = new Person();
}
```

**Lösung:**

```
+P Hans +PDef ?
```

**d.) (3 P.)** Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `Erz4` ?

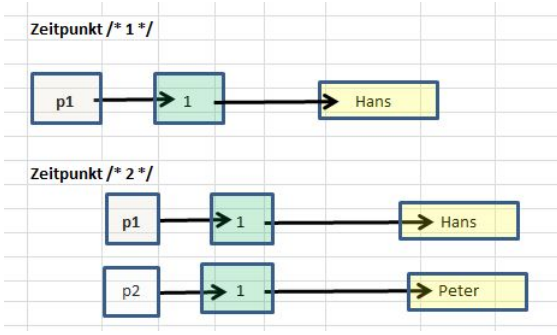
```
void Erz4() {
   unique_ptr<Person> p1(new Person("Hans"));
   if (p1 != nullptr)
   {
      unique_ptr<Person> p2(new Person("Paula"));
   }
   else
   {
      unique_ptr<Person> p3(new Person("Otto"));
   }
   datei << " Ende ";
}
```

**Lösung:**

```
+P Hans +P Paula -P Paula Ende -P Hans
```

Achtung: Die Ausgaben sind ab jetzt auf mehrere Teilfragen aufgeteilt!!!

Die folgende Zeichnung skizziert die Speicherbelegung zu den Zeitpunkten /* 1 */ und /* 2*/:



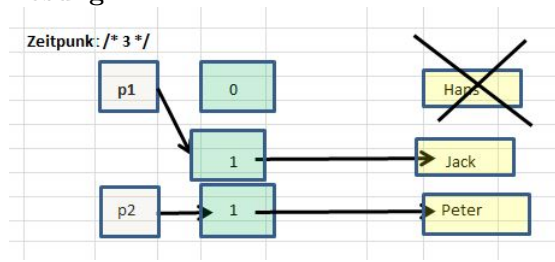

**e.) (2 P.)** Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `Erz5` (zwischen Funktionsanfang und nach Ausführung der Zeile /*1*/)?
**Lösung:**

```
+P Hans Ende
```

**f.) (2 P.)** Welche Ausgabe ergibt sich zwischen den Zeilen /*1*/ und /*2*/ (einschließlich /*2*/ und ausschließlich /*1*/)?
**Lösung:**

```
+PCop Peter Ende
```

**g.) (3 P.)** Skizzieren Sie entsprechend der obigen Zeichnung die Speicherbelegung zum Zeitpunkt /*3*/. Aus der Skizze sollte auch hervorgehen, wenn eine Instanz gelöscht wurde, z.B. durch Durchstreichen kennzeichnen.
**Lösung:**




**h.) (3 P.)** Welche Ausgabe ergibt sich zwischen den Zeilen /*2*/ und /*3*/ (einschließlich /*3*/ und ausschließlich /*2*/)?
**Lösung:**

```
+P Jack -P Hans Ende
```

**i.) (2 P.)** Welche Ausgabe ergibt sich zwischen den Zeilen /*3*/ und Ende /*4*/ (ausschließlich /*3*/)?
**Lösung:**

```
-P Peter -P Jack
```

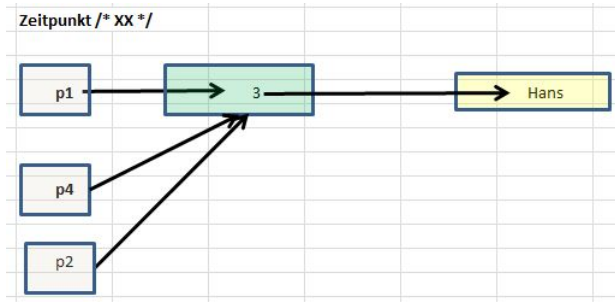
Das Objekt, auf das p1 verweist, wird natürlich als letztes zerstört, da p1 als erstes auf dem Stack erzeugt wurde.

```
void Erz5() {
   shared_ptr<Person> p1(new Person("Hans"));
   datei << " Ende\n"; /* 1 */
   shared_ptr<Person> p2 =
      make_shared<Person>(*p1);
   datei << " Ende\n "; /* 2 */
   p1 = make_shared<Person>("Jack");
   datei << " Ende\n"; /* 3 */
} /* 4 */
```

**j.) (3 P.)** Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `Erz6` (zwischen Funktionsanfang und nach Ausführung der Zeile /*1*/)?
**Lösung:**

```
+P Hans Stop
```

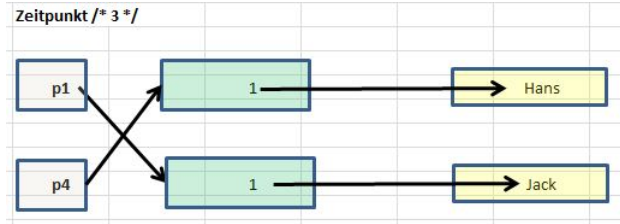
**k.) (3 P.)** Skizzieren Sie entsprechend der vorgegebenen Zeichnung die Speicherbelegung zum Zeitpunkt /*XX*/ bzw. zum Zeitpunkt /*YY*/ (je nach dem, welcher Pfad ausgeführt wird). Aus der Skizze sollte auch hervorgehen, wenn eine Instanz gelöscht wurde, z.B. durch Durchstreichen kennzeichnen.
**Lösung:**




**l.) (2 P.)** Welche Ausgabe ergibt sich zwischen den Zeilen /*1*/ und /*2*/ (einschließlich /*2*/ und ausschließlich /*1*/)?
**Lösung:**

```
Ende
```

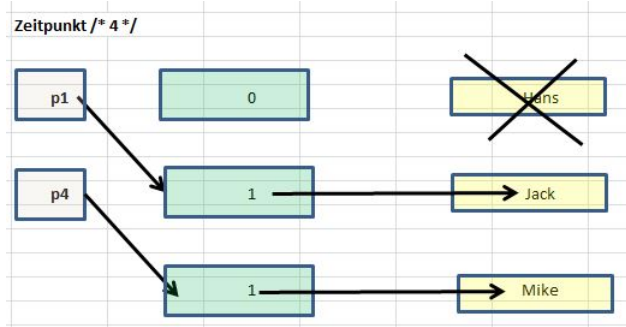
**m.) (3 P.)** Skizzieren Sie entsprechend der vorgegebenen Zeichnung die Speicherbelegung zum Zeitpunkt /*3*/. Aus der Skizze sollte auch hervorgehen, wenn eine Instanz gelöscht wurde, z.B. durch Durchstreichen kennzeichnen.
**Lösung:**




**n.) (2 P.)** Welche Ausgabe ergibt sich zwischen den Zeilen /*2*/ und /*3*/ (einschließlich /*3*/ und ausschließlich /*2*/)?
**Lösung:**

```
+P Jack Ende
```

**o.) (2 P.)** Skizzieren Sie entsprechend der vorgegebenen Zeichnung die Speicherbelegung zum Zeitpunkt /*4*/. Aus der Skizze sollte auch hervorgehen, wenn eine Instanz gelöscht wurde, z.B. durch Durchstreichen kennzeichnen.
**Lösung:**




**p.) (2 P.)** Welche Ausgabe ergibt sich zwischen den Zeilen /*3*/ und /*4*/ (einschließlich /*4*/ und ausschließlich /*3*/)?
**Lösung:**

```
+P Mike -P Hans Ende
```

**q.) (2 P.)** Welche Ausgabe ergibt sich zwischen den Zeilen /*4*/ und Ende /*5*/ (ausschließlich /*4*/)?
**Lösung:**

```
-P Mike -P Jack
```

```
void Erz6() {
   shared_ptr<Person> p1(new Person("Hans"));
   shared_ptr<Person> p4;
   datei << " Stop\n"; /* 1 */
   if (p1 != nullptr)
   {
      shared_ptr<Person> p2 = p1;
      p4 = p1; /* XX */
   }
   else
   {
      shared_ptr<Person> p3(p1); /* YY */
   }
   datei << " Ende\n "; /* 2 */
   p1 = make_shared<Person>("Jack");
   datei << " Ende\n"; /* 3 */
   p4 = make_shared<Person>("Mike");
   datei << " Ende\n"; /* 4 */
} /* 5 */
```

## Aufgabe 4 : Tief und flach kopieren

ca. 8 Punkte

Gegeben sei die folgende Klassenschnittstelle:

```
class Person {
public:
   Person(string na);
   ~Person();
   void AddFriend(Person* f);
   void PrintFriends() const;
private:
   string name;
   Person** friends;
   int friendsCnt;
   int friendsMax;
};
```

Implementieren Sie für die Klasse einen Zuweisungs-Operator (operator=).

Der folgende Code dient Ihnen hier lediglich als Hilfe. Sie können den Code ignorieren, wenn Sie sich zutrauen, den Zuweisungs-Operator zu implementieren. ohne die folgenden Hilfen zu nutzen. Die Methode AddFriend könnte allerdings hilfreich sein.

Die Methoden der Klasse sind wie folgt implementiert:

```
Person::Person(string na) {
   name = na;
   friends = nullptr;
   friendsCnt = 0;
   friendsMax = 0;
}

Person::~Person() {
   delete[] friends;
}
void Person::AddFriend(Person* f) {
   if (friendsCnt >= friendsMax) {
      Person** hlp = new Person*[friendsCnt + 1];
      for (int i = 0; i < friendsCnt; ++i) {
         hlp[i] = friends[i];
      }
      friendsMax = friendsCnt + 1;
      delete[] friends;
      friends = hlp;
   }
   friends[friendsCnt] = f;
   ++friendsCnt;
}
```

```
void Person::PrintFriends() const {
   if (friends) {
      datei << "Friends of " << name << ": ";
      for (int i = 0; i < friendsCnt; ++i) {
         datei << friends[i]->name << " ";
      }
   }
   else {
      datei << "Poor " << name
         << " has no friends!!!";
   }
   datei << endl;
}
```

Der folgende Testcode

```
void Test1() {
   Person ha("Hans");
   Person ka("Karl");
   Person ot("Otto");
   Person* p_an = new Person("Anna");
   ha.AddFriend(&ka);
   ha.AddFriend(&ot);
   ha.AddFriend(p_an);
   ha.PrintFriends();
   p_an->PrintFriends();
   delete p_an;
}
```

ergibt als Ausgabe:

```
Friends of Hans: Karl Otto Anna
Poor Anna has no friends!!!
```

Der folgende Testcode führt allerdings zu einem Absturz, da der Default- Zuweisungs-Operator nicht funktioniert.

```
void Test3() {
   Person ha("Hans");
   Person ot("Otto");
   ha.AddFriend(&ot);
   Person hans2("Hans2");
   hans2 = ha; // Problem
   Person mike("Mike");
   hans2.AddFriend(&mike);
   ha.PrintFriends();
   hans2.PrintFriends();
}
```

Nun sind Sie aber dran, mit der Implementierung des Zuweisungs-Operators, d.h. es geht um den fehlenden Code des folgenden Listings:

```
Person& Person::operator=(const Person& copy) {
   if (this != &copy) {
```

```
      /* Sie sind dran !!! */
```

```
   }
   return *this;
```

**Lösung:**

```
Person::Person(const Person& copy) {
   if (copy.friendsMax > 0 && copy.friends) {
      friends = new Person * [copy.friendsMax];
      for (int i = 0; i < copy.friendsCnt; ++i) {
         friends[i] = copy.friends[i];
      }
      friendsCnt = copy.friendsCnt;
      friendsMax = copy.friendsMax;
   }
   else {
      friendsCnt = 0;
      friendsMax = 0;
      delete[] friends;
      friends = nullptr;
   }
   name = copy.name;
}
```

Im folgenden Zuweisungsoperator erfolgt das Löschen erst nachdem sichergestellt ist, dass das Erzeugen des neuen Speichers auch wirklich geklappt hat. Wenn während des `new`-Aufrufs eine Ausnahme geworfen wird, bleibt der alte Zustand erhalten. Ganz wichtig ist, dass `delete[]` und nicht nur `delete` aufgerufen wird.

```
Person& Person::operator=(const Person& copy) {
   if (this != &copy) {
      if (copy.friendsMax > 0 && copy.friends) {
         Person** hlp = new Person * [copy.friendsMax];
         for (int i = 0; i < copy.friendsCnt; ++i) {
            hlp[i] = copy.friends[i];
         }
         delete[] friends;
         friendsCnt = copy.friendsCnt;
         friendsMax = copy.friendsMax;
         friends = hlp;
      }
      else {
         friendsCnt = 0;
         friendsMax = 0;
         delete[] friends;
         friends = nullptr;
      }
      name = copy.name;
   }
   return *this;
}
```

Eine alternative Lösung wäre AddFriend zu benutzen. Die Lösung ist aber weniger effizient, braucht aber weniger neuen Code.

```
Person::Person(const Person& copy) {
   friendsCnt = 0;
   friendsMax = 0;
   friends = nullptr;
   if (copy.friendsMax > 0 && copy.friends) {
      for (int i = 0; i < copy.friendsCnt; ++i) {
         AddFriend(copy.friends[i]);
      }
   }
   name = copy.name;
}
```

```
Person& Person::operator=(const Person& copy) {
   if (this != &copy) {
      delete[] friends; // nicht schoen, vorher zu loeschen
      friendsCnt = 0;
      friendsMax = 0;
      friends = nullptr;
      if (copy.friendsMax > 0 && copy.friends) {
         for (int i = 0; i < copy.friendsCnt; ++i) {
            AddFriend(copy.friends[i]);
         }
      }
      name = copy.name;
   }
   return *this;
}
```