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 2023/24:

Die verschiedenen Programmierparadigmen von C++

Hilfsmittel wie Bücher und Skripte und eigene Notizen sind erlaubt.
Die Nutzung eines Computers z.B. mit einer Programmierumgebung oder einem Compiler ist gar nicht erlaubt!
Austausch von Hilfsmitteln mit Kommilitonen ist **nicht** erlaubt !
Die Kommunikation mit Kommilitonen ist während der Klausur nicht erlaubt.
Anwesenheit von Handys, Smartphones etc. ist bei Klausurteilnehmern im Hörsaal nicht erlaubt.
Sie sind vor Beginn der Klausur am Dozentenpult abzugeben!

Bitte notieren Sie auf **allen** Blättern, die in die Bewertung eingehen sollen, Ihren Namen und Ihre Matrikelnummer.
Auf eine korrekte Anzahl der Blanks und Zeilenumbrüche braucht bei der Ausgabe nicht geachtet zu werden. Dafür werden keine Punkte vergeben.

Hinweis: In den folgenden Programmfragmenten wird die globale Variable `datei` verwendet. Hierfür kann der Einfachheit halber die Variable `cout` angenommen werden. Die Variable `datei` 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. Außerdem dürfen Sie annehmen, dass für alle Code-Fragmente die erforderlichen *include*-Anweisungen für C++-Header-Dateien erfolgt sind. Syntaxfehler sind allenfalls unabsichtlich in den Programmfragmenten enthalten.

Für einige Aufgabenteile ist ein Extrablatt zu verwenden; bitte mit Namen und Matrikelnummer beschriften. Die Größe der Lücken gibt keine Hinweise darauf, wie viel Ausgaben erfolgen. Sie dürfen auch alle Lösungen auf Extrablättern notieren.

Geplante Punktevergabe

Punktziel	Sonderpunkte	erreicht
Übungen:	XXX	
A1: 34 P.		
A2: 46 P.		
Summe 80+ SP.		

Aufgabe 1 : Objekterzeugung und -zerstörung

ca. 34 Punkte

Die Klasse `Mitarbeiter` hat folgende Schnittstelle:

```
class Mitarbeiter {
public:
    Mitarbeiter();
    Mitarbeiter(int id);
   ~Mitarbeiter();
    void SetWert(int w) { m_id = w; }

    // Kopier-Funktionalität
    Mitarbeiter(const Mitarbeiter& s2);
    Mitarbeiter& operator=(const Mitarbeiter& s2);
    // Verschiebe-Funktionalität
    Mitarbeiter(Mitarbeiter&& s2);
    void operator=(Mitarbeiter&& s2);
private:
    int m_id;
};
```

Die Implementierung der beiden Konstruktoren und des Destruktors ist:

```
Mitarbeiter::Mitarbeiter() {
   m_id = -1;
   datei << " +M " << m_id;
}
Mitarbeiter::Mitarbeiter(int id) {
   m_id = id;
   datei << " +M " << m_id;
}
Mitarbeiter::~Mitarbeiter() {
   datei << " -M " << m_id;
}
```

Eine Implementierung der Kopier- und Verschiebeoperatoren zeigen die beiden folgenden Listings:

```
Mitarbeiter::Mitarbeiter(const Mitarbeiter& cop) {
   m_id = cop.m_id;
   datei << " +MCop " << m_id;
}
Mitarbeiter&
Mitarbeiter::operator=(const Mitarbeiter& cop) {
   datei<<" op= old "<<m_id<< " new "<<cop.m_id;
   m_id = cop.m_id;
   return *this;
}
```

```
Mitarbeiter::Mitarbeiter(Mitarbeiter&& cop) {
   m_id = cop.m_id;
   datei << " +MMov " << m_id;
   cop.m_id = -2;
}
void Mitarbeiter::operator=(Mitarbeiter&& cop) {
   datei<<" move= this "<<m_id<<" cop "<<cop.m_id;
   m_id = cop.m_id;
   cop.m_id = -3;
}
```

a.) (ca. 4 P.) Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `f1` ?

```
void f1() {
   Mitarbeiter s1();
   Mitarbeiter s2(4);
   Mitarbeiter s3;
   datei << " Ende" << endl;/*1*/
}
```

(*— Lösung hier notieren. ——*)

b.) (ca. 3 P.) Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `f2` ?

```
void f2() {
   Mitarbeiter* p1 = new Mitarbeiter(4);
   unique_ptr<Mitarbeiter> p2(new Mitarbeiter(8));
   datei << " Ende" << endl;/*1*/
}
```

(*— Lösung hier notieren. ——*)

c.) (ca. 7 P.) Gegeben ist die Funktion `f3`:

```
void f3() {
   Mitarbeiter m1(5); /*1*/
   datei << "\n";
   Mitarbeiter m2(m1); /*2*/
   datei << "\n";
   Mitarbeiter m3(move(m1));
   datei << "\n"; /*3*/
}
```

Lösungen schrittweise auf diesem Arbeitsblatt notieren. Zu welcher Ausgabe führt ihr Aufruf bis /*1*/?

Ausgabe zwischen zwischen /*1*/ und /*2*/?

Ausgabe zwischen zwischen /*2*/ und /*3*/?

Ausgabe nach /*3*/?

d.) (ca. 8 P.) Gegeben ist die Funktion `f4`:

```
void funkR(Mitarbeiter& m) {
   datei << " funkR ";
   m.SetWert(51);
}
void funkW(Mitarbeiter m) {
   datei << " funkW";
   m.SetWert(31);
}
void f4() {
   Mitarbeiter m1(5);
   Mitarbeiter m2(50); /*1*/
   datei << "\n";
   funkR(m1);
   datei << "\n";/*2*/
   funkW(m2);
   datei << "\n"; /*3*/
}
```

Lösungen schrittweise auf diesem Arbeitsblatt notieren: Zu welcher Ausgabe führt ihr Aufruf bis /*1*/?

Ausgabe zwischen zwischen /*1*/ und /*2*/?

Ausgabe zwischen zwischen /*2*/ und /*3*/?

Ausgabe nach /*3*/?

e.) (ca. 6 P.) Gegeben ist die Funktion `f5`.

```
void f5(int arg) {
   Mitarbeiter m1(Mitarbeiter(8));
   datei << "\n";
   int a = arg * 4;    /*1*/
   if (a > 0) {
      Mitarbeiter m2 = m1;
      m2.SetWert(6);
   }
   else {
      Mitarbeiter m3[2];
      m3[0] = m1;
   }              /*2*/
   datei << "\n";
}
```

Ihr Aufruf mit dem **Wert `4`** führt zur folgenden Ausgabe:

```
+M 8
+MCop 8 -M 6
-M 8
```

Erklären Sie auf einem Extrablatt für jede der Zeilen, wodurch die jeweilige Ausgabe zustande kommt.

f.) (ca. 6 P.)
Zu welcher Ausgabe führt der Aufruf von `f5` mit dem Wert **minus `5`** zwischen /*1*/ und /*2*/?
.

Aufgabe 2 : Komplexe Objekterzeugung und -zerstörung

ca. 46 Punkte
Die Klasse `Hochschule` hat folgende Schnittstelle:

```
class Hochschule {
public:
   Hochschule();
   Hochschule(int a_anz);
   Hochschule(const Hochschule& s2);
   ~Hochschule();
   Hochschule& operator=(const Hochschule& s2);
   int GetAnz() const { return anz;}
private:
   Mitarbeiter dekan;
   Mitarbeiter * p_stud;
   int anz;
};
```

Die Implementierung der Konstruktoren und des Destruktors ist:

```
Hochschule::Hochschule() : dekan(114) {
   p_stud = nullptr;
   anz = 0;
   datei << "\n"; /* H1 */
   datei << " +H " << anz; /* H2 */
}
Hochschule::Hochschule(int a_anz) {
   datei << "\n"; /* H1 */
   dekan = Mitarbeiter(11);
   anz = a_anz; /* H2 */
   datei << "\n+H " << anz;
   if (anz > 0) {
      p_stud = new Mitarbeiter[a_anz]; /* H3 */
   }
   else {
      anz = 0;
      p_stud = nullptr;
   }
}
Hochschule::~Hochschule() {
   datei << " -H " << anz;
   delete[] p_stud;
}
```

Bei der Implementierung von Zuweisungsoperator und Kopierkonstruktor handelt es sich um die Default-Implementierung mit flacher Kopie.

a.) (ca. 5 P.) Welche Ausgabe (nach `datei`) ergibt der Aufruf der folgenden Funktion `g1` ? Siehe die Wiederholung des Codes in der rechten Spalte.

```
void g1() {
   Hochschule wolfenbuettel;
   datei << "\nEnde "; /*1*/
}
```

(*— Lösung hier notieren. ——*)

b.) (ca. 10 P.) Gegeben ist die Funktion `g2`.

```
void g2() {
   Hochschule wolfsburg(2);
   datei << "\nEnde ";/*1*/
}
```

Lösungen schrittweise auf diesem Arbeitsblatt notieren.
Zu welcher Ausgabe führt ihr Aufruf bis /*H1*/?

Ausgabe zwischen zwischen /*H1*/ und /*H2*/?

Ausgabe zwischen zwischen /*H2*/ und /*H3*/?

Ausgabe nach /*H3*/?

c.) (ca. 9 P.) Erklären Sie, warum die Funktion `g4` durch die fehlende bzw. falsche Implementierung des Zuweisungsoperators zu einem undefinierten Programmverhalten führt. Skizzieren Sie hierzu die Speicherbelegung auf Stack- und Heapspeicher am Ende der Funktion `g4` bei /*1*/ auf einem Extra-Blatt.

```
void g4() {
   Hochschule wolfenbuettel(3);
   Hochschule wolfsburg(2);
   datei << "\n vor copy\n";
   wolfsburg = wolfenbuettel;
   datei << " Ende" << endl;/*1*/
}
```

d.) (ca. 13 P.) Implementieren Sie nun einen korrekten Zuweisungsoperator. Notieren Sie Ihre Lösung auf einem Extra-Blatt. Ihre Implementierung wird sehr wahrscheinlich mindestens eine Verzweigung (`if`-Anweisung) benötigen.

e.) (ca. 9 P.) Implementieren Sie nun verschiedene Unit-Tests, die zeigen, dass Sie Ihren neuen Zuweisungsoperator auf korrekte Funktionalität prüfen. Nutzen Sie die Methode `GetAnz` sinnvoll. Ihre Tests dürfen abstürzen, wenn die fehlerhafte Default-Implementierung des Zuweisungsoperator vorliegen würde. Sofern Ihre Implementierung des Zuweisungsoperators Verzweigungen benötigt, implementieren Sie verschiedene Tests, sodass zumindest **alle** Pfade und Pfadkombinationen im Zuweisungsoperator durchlaufen werden. Beachten Sie: Tests liefern im Erfolgsfall `true` zurück, sonst `false` und verschiedene Tests sind unabhängig voneinander ausführbar.

Zur Vermeidung des Umblätterns folgt hier nochmals die Implementierung der Schnittstelle der Klasse `Hochschule`:

```
Hochschule::Hochschule() : dekan(114) {
   p_stud = nullptr;
   anz = 0;
   datei << "\n"; /* H1 */
   datei << " +H " << anz; /* H2 */
}
Hochschule::Hochschule(int a_anz) {
   datei << "\n"; /* H1 */
   dekan = Mitarbeiter(11);
   anz = a_anz; /* H2 */
   datei << "\n+H " << anz;
   if (anz > 0) {
      p_stud = new Mitarbeiter[a_anz]; /* H3 */
   }
   else {
      anz = 0;
      p_stud = nullptr;
   }
}
Hochschule::~Hochschule() {
   datei << " -H " << anz;
   delete[] p_stud;
}
```