Probeklausur **Programmentwicklung 2**

Nachname	Vorname
Matrikelnummer	Studiengang

Hinweise:

- Bearbeitungszeit: 120 Minuten.
- Die Bearbeitung darf nur mit dokumentenechten Stiften erfolgen. Verwenden Sie keinen Bleistift. Verwenden Sie keinen Stift, der in Rot schreibt.
- Bei unterschiedlichen abgegebenen Lösungen für eine Aufgabe, wird die Aufgabe nicht bewertet. Antworten, die der Korrektor nicht lesen kann, werden nicht bewertet.
- Verwenden Sie keine Korrekturflüssigkeiten (*TippEx*) oder Tintenkiller. Nicht zu bewertende Lösungsteile sind durchzustreichen.
- Erlaubte Hilfsmittel: ein nicht-programmierbarer Taschenrechner sowie alle weiteren nicht-technischen Hilfsmittel.
- Mobiltelefone, Smartwatches und ähnliche Geräte sind ausgeschaltet in der Tasche zu verwahren. Die Verwendung eines solchen Gerätes wird als Täuschungsversuch gewertet.

Durch Unterschrift bestätigen Sie, dass Sie die Hinweise zur Bearbeitung der Klausur gelesen und verstanden haben, und sie als Bestandteile der Prüfungsbedingungen anerkennen.

Unterschrift

Aufgabe:	1	2	3	4	5	6	7	8	Σ	**Note**
Punkte:	20	20	20	20	20	20	20	20	160	
Erreicht:										

1. Prüfer: Dr. Michael Gref

2. Prüfer*in

1 Aufgabe (Fehlersuche)

Die nachfolgenden Teilaufgaben enthalten Codeausschnitte, die jeweils einen oder mehrere Fehler beinhalten. Finden und korrigieren jeweils Sie den oder die Fehler unter Angabe der entsprechenden Zeilennummern.

1.1 Teilaufgabe [10]

Gegeben sei das folgende `C++`-Programm, das eine vereinfachte Version der Addition von Brüchen (ohne Kürzen) mittels Operatorüberladung realisieren soll:

```
1  #include<iostream>
2
3  class Bruch {
4    private:
5    int _zaehler;
6    int _nenner;
7    public:
8    Bruch(int zaehler, int nenner): _zaehler(zaehler), _nenner(
       nenner) {
9      if(_nenner == 0) {
10       throw std::invalid_argument("Nenner kann nicht Null sein.");
11     }
12   }
13
14   Bruch add(const Bruch& x) {
15     return Bruch(_zaehler + x._zaehler, _nenner * x._nenner);
16   }
17
18   void print() {
19     std::cout << _zaehler << "/" << _nenner << std::endl;
20   }
21 };
22
23 int main() {
24   try {
25     Bruch f1(1, 2);
26     Bruch f2(1, 3);
27     Bruch f3 = f1 + f2;
28     f3.print();
29   } catch(const std::exception& e) {
30     std::cerr << e.what() << '\n';
31   }
32   return 0;
33 }
```

Die erwartete Ausgabe ist `5/6`.

1.2 Teilaufgabe [10]

Gegeben sei das folgende C++-Programm:

```
#include <iostream>
#include <string>

class Auto {
  private:
  std::string marke;
  int kilometerstand;

  protected:
  Auto(std::string m, int k) : marke(m), kilometerstand(k) {}

  void fahre(int km) const {
    kilometerstand += km;
  }

  std::string getMarke() {
    return marke;
  }

  int getKilometerstand() {
    return kilometerstand;
  }
};

int main() {
  Auto meinAuto("Tesla", 0);
  meinAuto.fahre(100);
  std::cout << "Marke: " << meinAuto.getMarke() << ",
   Kilometerstand: " << meinAuto.getKilometerstand() << std::endl;
  return 0;
}
```

Die erwartete Ausgabe ist `Marke: Tesla, Kilometerstand: 100`.

2 Aufgabe (Programmverständnis)

2.1 Teilaufgabe [10]

Gegeben sei das folgende C++-Programm:

```
#include<iostream>

class Probe {
  public:
  Probe() {
    std::cout << "X ";
  }
  ~Probe() {
    std::cout << "Z ";
  }
  Probe fun1(Probe &p) {
    return p;
  }

};

void fun2(Probe &p) {
  Probe z = p;
}

int main() {
  Probe p1;
  p1 = p1.fun1(p1);
  fun2(p1);
  Probe p2 = p1;
  return 0;
}
```

Wie ist die Ausgabe in der Konsole?

2.2 Teilaufgabe [10]

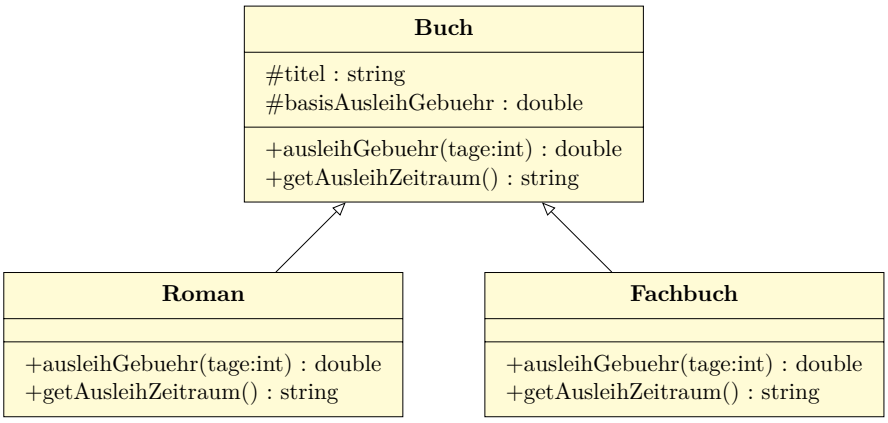
Gegeben sei das folgende C++-Programm:

```
#include<iostream>

template<typename T>
bool equal(T a, T b) {
  return a == b;
}

int main() {
  std::cout << equal<int>(3, 4) << "\n";
  std::cout << equal<double>(3, 3.0) << "\n";

  int a=1, b=1;
  std::cout << equal<int*>(&a, &b) << "\n";
  return 0;
}
```

Ist das gegebene Programm funktional oder beinhaltet es Fehler? Wie ist die Ausgabe des Programms für alle Zeilen, die keinen Fehler beinhalten?

3 Aufgabe (Polymorphie) [20]

In der folgenden Darstellung eines Klassendiagramms wird ein Ausschnitt aus einer Software zur Verwaltung einer Bibliothek angezeigt. Jedes Buch hat spezielle Ausleihbedingungen, die die Ausleihgebühren bestimmen.



Die Methoden `ausleihGebuehr` und `getAusleihZeitraum` der jeweiligen Klassen sind wie folgt aufgebaut:

- Buch: : ausleihGebuehr = basisAusleihGebuehr $*$ tage
  AusleihZeitraum ist `"2 Monate"`
- Roman: ausleihGebuehr = basisAusleihGebuehr $*$ tage. Maximal jedoch 5 Euro.
  AusleihZeitraum ist `"1 Monat"`
- Fachbücher haben eine feste Gebühr von 2€, unabhängig von der Anzahl der Tage.
  AusleihZeitraum ist `"2 Wochen"`

Stellen Sie mittels dynamischer Bindung sicher, dass die Methoden der zugehörigen Objekte verwendet werden, auch wenn auf Objekte der Unterklassen mittels Zeigern vom Typ der Oberklasse zugegriffen wird.

Geben Sie eine Implementierung in `C++` an, die die Klassen und Methoden entsprechend den Anforderungen implementiert.

4 Aufgabe (Ein-/Ausgabe) [20]

Gegeben sei ein Array von Mitarbeiter-Objekten, für die Verwaltung der Daten der Mitarbeiter, die in einem Unternehmen arbeiten. Jedes Mitarbeiter-Objekt hat eine nummerische ID, einen Namen, eine Abteilung und ein Gehalt. Beispiel:

```
Mitarbeiter mitarbeiter[] = {
  {12345, "Max Mustermann", "Vertrieb", 50000.0},
  {23456, "Erika Musterfrau", "IT", 55000.0},
  {34567, "Otto Normalverbraucher", "Marketing",
     48000.0},
  {45678, "Anna Normal", "Finanzen", 52000.0}
};
```

1. Definieren Sie die Mitarbeiter-Klasse, so dass diese zu dem obigen Programmfragment kompatibel ist.
2. Schreiben Sie eine `C++`-Funktion `void storeMitarbeiterData(string filename , Mitarbeiter mitarbeiter[], int size)`, die den Dateipfad, ein Mitarbeiter-Array und dessen Größe als Argumente übergeben bekommt. Die Funktion soll die Mitarbeiterinformationen in eine Datei schreiben. Die einzelnen Werte sollen mittels Komma voneinander getrennt sein. Beispiel:

```
12345,Max Mustermann,Vertrieb,50000.0
23456,Erika Musterfrau,IT,55000.0
34567,Otto Normalverbraucher,Marketing,48000.0
45678,Anna Normal,Finanzen,52000.0
```

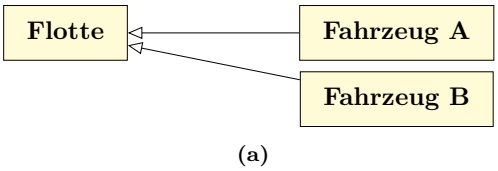
   Hinweise: Falls die Datei nicht geöffnet werden konnte, soll eine Hinweis in die Standardausgabe geschrieben werden. Geben Sie alle notwendigen Header an. Achten Sie ggf. auf die Formatierung der Ausgabe.

5 Aufgabe (Klassenbeziehungen)

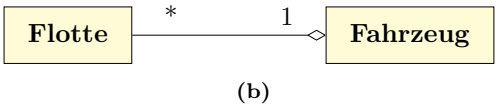
In den folgenden Teilaufgaben, ist jeweils eine Aussage sowie verschiedene UML-Klassendiagramme gegeben. **Begründen** Sie kurz(!) für jedes Diagramm, ob eine Realisierung auf Basis dieses Diagramms geeignet ist, die Zusammenhänge der gegebene Aussage als Klassenbeziehung zu modellieren oder nicht.

5.1 Teilaufgabe [10]

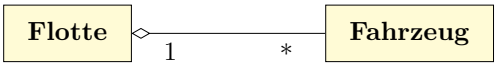
Aussage: Die Fahrzeugflotte eines Unternehmens besitzt mehrere Fahrzeuge. Ein Fahrzeuge gehört zu genau einer Fahrzeugflotte.



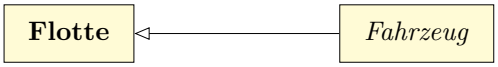
(a)



(b)



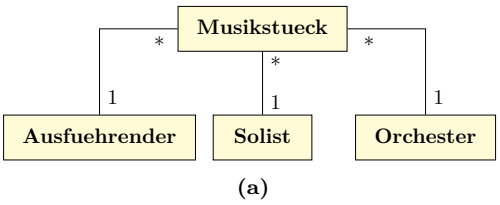
(c)



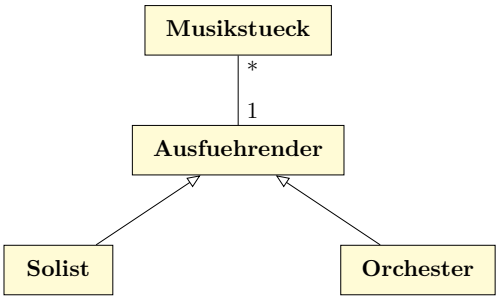
(d)

5.2 Teilaufgabe [10]

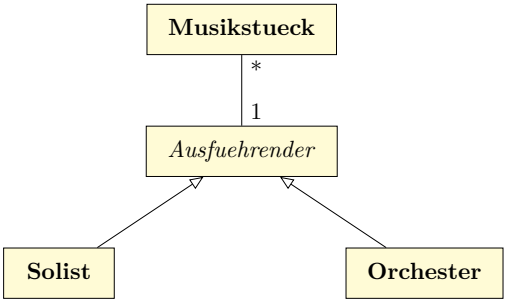
Aussage: Jedes Musikstück wird entweder ausschließlich von einem Solisten, von einem Orchester oder von einem anderen Ausführenden gespielt.



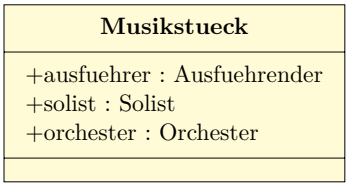
(a)



(b)



(c)



(d)

6 Aufgabe (Standard Template Library) [20]

6.1 Teilaufgabe Functor [10]

Schreiben Sie einen Functor, der zu dem folgenden (und ähnlichen) Programmfragment kompatibel ist, sodass jedes Element des Vektors durch eine gegebene Zahl geteilt wird (im Beispiel 6). Der Functor sollte außerdem überprüfen, ob eine Division durch Null erfolgt und in diesem Fall einen Exception vom Typ `invalid_argument` werfen.

```
std::vector<int> numbers_1 = {12, 24, 36, 48, 60, 72, 84, 96};
try {
  Divide divideBySix(6);
  std::for_each(numbers_1.begin(), numbers_1.end(), divideBySix);
} catch (std::invalid_argument& e) {
  std::cout << e.what() << std::endl;
}
```

6.2 Teilaufgabe Predicates [10]

Schreiben Sie ein Predicate, das zu dem folgenden (und ähnlichen) Programmfragment kompatibel ist, sodass der letzte Wert im Vektor zurückgeliefert wird, der eine Quadratzahl ist:

```
1 std::vector<int> numbers_2 = {25, 36, 49, 64, 81, 100, 121, 143};
2 IsSquareNumber isSquareNumber;
3 std::vector<int>::reverse_iterator it2 = std::find_if(numbers_2.
     rbegin(), numbers_2.rend(), isSquareNumber);
```

Hinweis: Sie dürfen für die Berechnung `double sqrt(doble x)` der `<cmath>` verwenden.

7 Aufgabe (Entwurfsmuster) [20]

Sie arbeiten für ein Unternehmen, das sich auf die Herstellung und Programmierung von Robotern spezialisiert hat. Jeder Roboter hat verschiedene Bewegungsmodi, die sich je nach Situation und Umgebung ändern und aktiviert werden können:

- *Gehen*: `void walk()`
- *Laufen*: `void run()`
- *Schwimmen*: `void swim()`

Ihr Ziel ist es, unter Einsatz objektorientierter Programmierung eine flexible Softwarearchitektur zu entwerfen, die es erlaubt, auch in Zukunft neue Bewegungsmodi zu einem Roboter hinzuzufügen, ohne dass die Roboterklasse geändert werden muss. Darüber hinaus sollte jeder Roboter zur Laufzeit in der Lage sein, seinen Bewegungsmodus zu ändern.

Es ist zu jeden Zeitpunkt immer nur ein Bewegungsmodus aktiv, der mittels der Methode `void move(...)` der Roboterklasse aufgerufen werden wird.

1. Welches der aus der Vorlesung bekannten Entwurfsmuster würde diese Anforderungen am besten erfüllen und warum (Begründung!)?
2. Skizzieren Sie ein UML-Klassendiagramm für die entsprechende Lösung, und erklären Sie kurz(!) die Rolle jeder Klasse in Ihrem Diagramm.
3. Gehen Sie davon aus, dass die o.g. Methoden für Gehen, Laufen und Schwimmen, gemäß Ihres Architekturvorschlags, bereits implementiert sind. Geben Sie den `C++`-Code für die Methode `void move(...)` der Roboterklasse an.

8 Aufgabe (Verteilte Systeme) [20]

Gegeben sei die folgende C++-Implementierung eines Clients in einer Server-Client-Anwendung, die via Sockets Daten in Form eines Strings vom Client zum Server sendet.

```
#include <iostream>
#include <cstring> // Needed for memset
#include <sys/socket.h>
#include <arpa/inet.h> // For inet_addr
#include <unistd.h>

#define MAX_BUFFER_SIZE 1024
#define PORT 8081

int main() {
  int sock;
  struct sockaddr_in server;
  char message[MAX_BUFFER_SIZE];

  // Create socket
  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock == -1) {
    std::cerr << "Could not create socket\n";
    return 1;
  }

  server.sin_addr.s_addr = inet_addr("127.0.0.1");
  server.sin_family = AF_INET;
  server.sin_port = htons(PORT);

  // Connect to server
  if (connect(sock, (struct sockaddr*)&server, sizeof(server)) <
   0) {
    std::cerr << "Connect failed\n";
    return 1;
  }

  std::cout << "Connected\n";

  // Communicate with the server
  std::cout << "Enter a message: ";
  std::cin.getline(message, MAX_BUFFER_SIZE);
  if (send(sock, message, strlen(message), 0) < 0) {
    std::cerr << "Send failed\n";
    return 1;
  }

  std::cout << "Closing connection...\n";
```

```
43    close(sock);
44
45    return 0;
46  }
```

1. Wir wollen zur Steigerung der Effizienz der Kommunikation nun nicht mehr ASCII-Zeichen sondern ganze Zahlen (`Integer`) im Binärformat versenden. Passen Sie den Client entsprechend an, dass nun ein Integer-Wert anstelle eines Strings versendet wird.
   Hinweis: Sie müssen nicht den bestehend Quellcode abschreiben. Geben Sie Zeilennummern an, um kenntlich zu machen, wo Sie welche Änderungen vornehmen.
2. Erläutern Sie kurz, welche Änderungen auf Seiten des Servers notwendig sind und welche Aspekte beim Versand von Binärdaten noch zu beachten sind, um die gewünschte Funktionalität zu realisieren.