ulm university universität ulm

Klausurdeckblatt

Name der Prüfung:	Objektorientierte Programmierung mit C++		
Datum und Uhrzeit:	2. März 2017, 14-16 Uhr	Prüfer:	Dr. Andreas F. Borchert
Bearbeitungszeit:	120 Min.	Institut:	Numerische Mathematik

Vom Prüfungsteilnehmer auszufüllen:

Name: _____ Studiengang: _____ Matrikelnummer:

Vorname: _____ Abschluss: _____ _____

_____

Datum, Unterschrift des Prüfungsteilnehmers

Hiermit erkläre ich, dass ich prüfungsfähig bin. Sollte ich aufgrund fehlender Anmeldung über das Hochschulportal oder über das Studiensekretariat nicht auf der Liste der angemeldeten Studierenden aufgeführt sein, dann nehme ich hiermit zur Kenntnis, dass diese Prüfung nicht gewertet werden wird.

Hinweise zur Prüfung:
siehe nächstes Blatt

Erlaubte Hilfsmittel:
Bis zu fünf handgeschriebene Blätter.

Bitte dieses Feld für den Barcode freilassen!

Vom Prüfer auszufüllen:

Erreichte Punkte: _____

Note: _____

_____

Datum, Unterschrift Prüfer (Dr. Andreas F. Borchert)

- Prüfen Sie zu Beginn, ob Ihre Klausur vollständig ist beginnend von der Aufgabe 1 auf Seite 0(!) bis zur letzten Seite 22.
- Für Ihre Lösungen verwenden Sie bitte den freigelassenen Platz nach der Aufgabenstellung, die gegenüberliegende Seite der jeweiligen Aufgabe oder die angehängte leere Seite unter Angabe der Aufgabennummer.
- Nennen Sie möglichst alle Annahmen, die Sie gegebenenfalls für die Lösung einer Aufgabe treffen!
- Sofern nichts anderes angegeben ist, können Sie bei den Programmier-Aufgaben auf die Angabe der notwendigen *#include*-Anweisungen verzichten.
- Wenn es bezüglich der Aufgabenstellung Unklarheiten gibt, dann scheuen Sie sich bitte nicht, jemanden von der Aufsicht zu befragen.
- Wenn wir während der Klausur feststellen, dass eine Aufgabenstellung missverständlich ist, werden wir an der Tafel einen klärenden Hinweis für alle sichtbar hinschreiben.
- Hinweise zu der Bewertung:
  - Punktzahlen für Teilaufgaben werden nur ganzzahlig vergeben. Wenn notwendig, wird abgerundet.
  - Bei Aufgaben, bei denen Antworten anzukreuzen sind, löscht ein falsches Kreuz ein korrektes Kreuz aus. Negative Punkt für Teilaufgaben werden jedoch nicht vergeben, schlimmstenfalls sind es nur 0 Punkte.

Nr	Max	Bewertung	
1	**8**	xxxxx	
(a)	4		xxxxx
(b)	4		xxxxx
2	**18**	xxxxx	
(a)	1		xxxxx
(b)	1		xxxxx
(c)	1		xxxxx
(d)	1		xxxxx
(e)	6		xxxxx
(f)	2		xxxxx
(g)	3		xxxxx
(h)	3		xxxxx
3	**10**	xxxxx	
4	**10**	xxxxx	
(a)	4		xxxxx
(b)	2		xxxxx
(c)	4		xxxxx
5	**17**	xxxxx	
(a)	3		xxxxx
(b)	3		xxxxx
(c)	3		xxxxx
(d)	2		xxxxx
(e)	6		xxxxx

Nr	Max	Bewertung	
6	**14**	xxxxx	
(a)	2		xxxxx
(b)	2		xxxxx
(c)	2		xxxxx
(d)	2		xxxxx
(e)	2		xxxxx
(f)	4		xxxxx
7	**10**	xxxxx	
(a)	1		xxxxx
(b)	3		xxxxx
(c)	2		xxxxx
(d)	2		xxxxx
(e)	2		xxxxx
8	**13**	xxxxx	
(a)	2		xxxxx
(b)	2		xxxxx
(c)	2		xxxxx
(d)	1		xxxxx
(e)	2		xxxxx
(f)	4		xxxxx
Summe	**100**		

0

Aufgabe 1 **(8 Punkte)**

(a) 4 Punkte

Schreiben Sie ein übersetzbares C++-Programm, das eine ganze Zahl einliest und dessen Quadrat ausgibt. Bitte geben Sie bei dieser Aufgabe auch die **#include**-Anweisungen mit an.

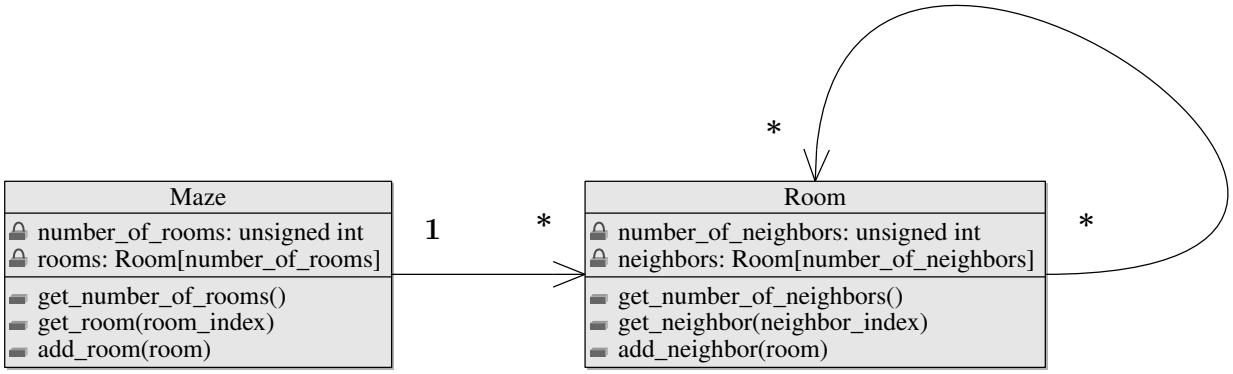
Lösung:

(b) 4 Punkte

Wählen Sie einen der in Ihrer Lösung verwendeten Operatoren aus: Welche Datentypen haben die Operanden und welcher Datentyp wird von dem Operator zurückgeliefert?

Lösung:

Aufgabe 2 **(18 Punkte)** Umsetzung von Klassendiagrammen in C++



(a) 1 Punkte

Charakterisieren Sie kurz die Beziehung zwischen den Klassen *Maze* und *Room*.

Lösung:

(b) 1 Punkte

Welche der beiden Klassen ist für die Beziehung zwischen *Maze* und *Room* verantwortlich?

Lösung:

(c) 1 Punkte

Kann ein Objekt der Klasse *Room* mit mehreren Objekten der Klasse *Maze* in Beziehung stehen?

Lösung:

(d) 1 Punkte

Ist *number_of_rooms* in der Klasse *Maze* öffentlich zugänglich?

Lösung:

(e) 6 Punkte

Schreiben Sie die vollständige Header-Datei für *Room.hpp* mitsamt allen notwendigen Präprozessor-Direktiven, in der die Klasse *Room* wie im Klassendiagramm beschrieben vollständig deklariert wird. D.h. dass bei den Methoden nur die Signaturen anzugeben sind und die jeweilige Implementierung wegfällt.

Lösung:

(f) 2 Punkte

Welche Präprozessor-Anweisungen benötigen Sie zu Beginn in *Room.cpp*?

Lösung:

(g) 3 Punkte

Implementieren Sie die Methode *get_number_of_neighbors*.
Lösung:

(h) 3 Punkte

Implementieren Sie die Methode *add_neighbor*.
Lösung:

Aufgabe 3 **(10 Punkte)** Erstellung eines Klassendiagramms
Erstellen Sie ein UML-Klassendiagramm zu folgender Beschreibung: Zu einem Blog gehören beliebig viele Beiträge und beliebig viele Autoren. Jeder Beitrag gehört nur zu einem Blog, aber ein Autor kann an mehr als einem Blog mitwirken. Jeder Beitrag hat genau einen Autor und beliebig viele Kommentare. Jeder Kommentar gehört genau zu einem Beitrag und stammt genau von einem Autor. Ein Autor kann beliebig viele Beiträge und Kommentare verfassen.
Bitte legen Sie im Klassendiagramm die Verantwortlichkeiten für folgende Navigationswege fest: Ausgehend von einem Blog sollte es möglich sein, an sämtliche Beiträge und sämtliche darin aktiv gewordene Autoren zu gelangen. Ausgehend von einem Autor sollten alle von ihm verfassten Beiträge und Kommentare zu finden sein. Ausgehend von einem Beitrag sollte der Autor und alle Kommentare abfragbar sein. Zu jedem Kommentar sollte der Autor nachgefragt werden können.
Zeichnen Sie nur die Klassen und ihre Beziehungen mitsamt den Komplexitätsgraden. Die jeweiligen Verantwortlichkeiten für die Beziehungen sollten aus dem Diagramm hervorgehen. Methoden oder Datenfelder müssen Sie nicht angeben.

Aufgabe 4 **(10 Punkte)** Iteratoren

(a) 4 Punkte

Definieren Sie den Begriff Iterator. Nennen Sie insbesondere die wesentlichen Eigenschaften, die einen Iterator in C++ ausmachen. Welche Operatoren werden unterstützt? Es genügt, einen einfachen vorwärts laufenden Iterator (*ForwardIterator*) zu beschreiben.

Lösung:

(b) 2 Punkte

Sie haben einen sich an die üblichen Konventionen haltenden Container *c*, bei dem zu einem Zeitpunkt *c.begin*()== *c.end*() gilt. Was können Sie dann über den Container aussagen?

Lösung:

(c) 4 Punkte

Gegeben sei der bereits mit Werten gefüllte Vektor *a*:

std::*vector*<**double**> *a*;

Schreiben Sie folgenden Programmtext so um, dass auf die explizite Indizierung des Vektors verzichtet wird, um sämtliche Werte des Vektors zu verdoppeln:

```
for (std::size_t i = 0; i < a.size(); ++i) {
    a[i] *= 2;
}
```

Lösung:

Aufgabe 5 **(17 Punkte)** Dynamische Datenstrukturen

(a) 3 Punkte

In welche drei Speicherbereiche können in C++ Variableninhalte abgelegt werden?

Lösung:

(b) 3 Punkte

Gegeben sei folgende Klasse *Person*:

```
class Person {
    public:
        Person(const std::string& name) : name(name) {
        }
        const std::string& get_name() const {
            return name;
        }
    private:
        std::string name;
};
```

Zeigen Sie an einem Programmtextbeispiel, wie in einem Block zwei *Person*-Objekte erzeugt werden, die in zwei verschiedenen Speicherbereichen leben, und wie diese Objekte bei oder vor dem Verlassen des umgebenden Blocks wieder korrekt freigegeben werden. Bei beiden Objekten sollte kurz kommentiert werden, in welche der drei Speicherbereiche sie leben.

Lösung:

(c) 3 Punkte

Was ist ein Speicherleck? Zeigen Sie an einem Programmtextbeispiel, wie ein Speicherleck entstehen kann.

Lösung:

(d) 2 Punkte

Erklären Sie kurz den Sinn der *std*::*shared_ptr*-Template-Klasse.

Lösung:

(e) 6 Punkte

Definieren Sie eine Klasse *Pedigree*, bei der jedes Objekt eine Person repräsentiert, die einen Namen hat (analog zu oben) und, sofern bekannt, *std*::*shared_ptr*-Zeiger auf *Pedigree*-Objekte liefert, die die Mutter bzw. den Vater repräsentieren.

Lösung:

Aufgabe 6 **(14 Punkte)** Templates

(a) 2 Punkte

Werden Templates in C++ zur Übersetzzeit oder zur Laufzeit instantiiert?

Lösung:

(b) 2 Punkte

Sie haben ein Template, das Sie mit unterschiedlichen Parametern mehrfach instantiieren. Führt das zu einem höheren generierten Code-Umfang oder bleibt dieser davon unberührt?

Lösung:

(c) 2 Punkte

Was ist der Unterschied zwischen dem Inclusion- und dem Cfront-Modell? Welches davon wird durch den aktuellen Standard (C++11) garantiert unterstützt?

Lösung:

(d) 2 Punkte

Gegeben sei folgende Template-Klasse:

```
#include <cassert>
template<typename Item, unsigned int size>
class Array {
    public:
        Array() {}
        std::size_t get_size() const { return size; }
        Item& operator[](std::size_t index) {
            assert(index < size);
            return a[index];
        }
    private:
        Item a[size];
};
```

Deklarieren Sie auf Basis dieser Template-Klasse *Array* ein Array *a* mit 20 Elementen des Typs **int**.

Lösung:

(e) 2 Punkte

Wäre es auch möglich, auf Basis dieser Template-Klasse ein Array mit Elementen des Typs *Person* (Aufgabe 5b, Seite 10) zu deklarieren? Wenn nein, warum nicht?

Lösung:

(f) 4 Punkte

Schreiben Sie ein Funktions-Template *initialize_array*, mit dem ein auf der obigen Template-Klasse *Array* basierendes Array mit Hilfe eines Lambda-Ausdrucks initialisiert werden kann, der jeweils den aktuellen Index als Parameter erhält. Sei *a* das oben deklarierte Array, dann sollte es beispielsweise mit

initialize_array(*a*, [](*std*::*size_t i*){ **return** $i * i$; });

möglich sein, das Array mit den Quadratzahlen $0^2, \ldots, 19^2$ zu füllen.

Lösung:

Aufgabe 7 **(10 Punkte)** STL

(a) 1 Punkte
Nennen Sie eine beliebige STL-Container-Klasse aus dem C++11-Standard Ihrer Wahl.

Lösung:

(b) 3 Punkte
Beschreiben Sie mindestens drei wesentliche Eigenschaften des Containers. Gehen Sie dabei insbesondere auf den Zugriff und das Hinzufügen von Elementen ein.

Lösung:

(c) 2 Punkte
Welche Komplexität hat das Hinzufügen eines Objekts in dem genannten Container, wenn bereits n Objekte enthalten sind?

Lösung:

(d) 2 Punkte
Nennen Sie ein Anwendungsszenario, für das der von Ihnen gewählte Container *nicht* geeignet ist.

Lösung:

(e) 2 Punkte
Welche STL-Container-Klasse wäre geeignet, um Schlüssel-Werte-Paare unterzubringen, bei denen ein Schlüssel mehrfach vorkommen kann, bei denen es auf eine schnelle Suche über den Schlüssel ankommt und bei dem die Schlüssel sortiert angeordnet sind? Welche Komplexität hat hier das Hinzufügen eines Paares, wenn bereits n Paare enthalten sind?

Lösung:

Aufgabe 8 **(13 Punkte)** Traits

Gegeben sei folgendes Programm:

```
#include <iostream>
#include <list>

template <typename T>
struct AverageTraits {
    using Value = T;
};

template <typename ForwardIterator,
    typename AT = AverageTraits<typename ForwardIterator::value_type> >
struct Average {
    using Value = typename AT::Value;
    static Value average(ForwardIterator begin, ForwardIterator end) {
        Value sum = Value(); Value count = Value();
        for (ForwardIterator it = begin; it != end; ++it) {
            sum += *it; ++count;
        }
        return sum / count;
    }
};

template <typename ForwardIterator>
inline typename AverageTraits<typename ForwardIterator::value_type>::Value
average(ForwardIterator begin, ForwardIterator end) {
    return Average<ForwardIterator>::average(begin, end);
}

int main() {
    std::list<int> values;
    for (int i = 1; i < 3; ++i) {
        values.push_back(i);
    }
    std::cout << "avg = " << average(values.begin(), values.end()) << std::endl;
}
```

(a) 2 Punkte
Was wird unter dem Begriff *Traits* in C++ verstanden?

Lösung:

(b) 2 Punkte
Mit welchem Datentyp für den Template-Parameter *ForwardIterator* wird in *main()* die Template-Funktion *average* instantiiert?

Lösung:

(c) 2 Punkte
Welchem Datentyp entspricht dann *ForwardIterator*::*value_type*?

Lösung:

(d) 1 Punkte
Wenn in Folge des Aufrufs von *average* die Template-Klasse **struct** *Average* instantiiert wird, welchen Datentyp erhält dann *Value*?

Lösung:

(e) 2 Punkte
Welche Ausgabe erzeugt das Programm?

Lösung:

(f) 4 Punkte
Wie könnte die Template-Konstruktion rund um *average* ergänzt werden, so dass die Anwendung in *main* ein geeigneteres Resultat liefert? Hinweis: Es ist nur eine Deklaration hinzuzufügen. Welche Ausgabe wird dann erzeugt?

Lösung: