Objekte / Objektorientierte Programmierung Klausur WS 2019/2020 (mit Lösungen und Bewertungsschema)

Aufg. Nr.	Text	Antwort/Lösung	Pkt
1.	a) Natürlich sprachliche Anforderungen bei einer objektorientierten Analyse: Welche Rollen spielen die Satzbestandteile? Ordnen Sie zu, indem Sie die zusammengehörenden Begriffe durch **Linien** verbinden: Substantive O O sind Kandidaten für Klassen Adjektive O O sind Kandidaten für Methoden Verben O O sind Kandidaten für Attribute b) Erklären Sie den Begriff „vollständige Überdeckung“ von Klassen anhand von Beispielobjekten. Verwenden Sie die 4 Klassen „Tulpe“, “Rose“, „Blume“ und „Sonnenblume“ entweder mit Worten oder mit einer Grafik. c) Was ist der Unterschied zwischen einer konkreten Klasse und einer abstrakten Klasse?	a) Je richtiger Zuordnung (Verbindungslinie) 1 P 3P • Substantive sind Kandidaten für Klassen • Adjektive sind Kandidaten für Attribute (Mitgliedsvariablen) • Verben sind Kandidaten für Methoden (Mitgliedsfunktionen) b) Alle Instanzen sind von abgeleiteten Klassen, dann ist die Basisklasse (Superklasse, Oberklasse) vollständig durch ihre abgeleiteten Klassen (Subklassen, Unterklassen) überdeckt. 4P Jede Blume ist entweder eine Rose oder eine Sonnenblume oder eine Tulpe, es ist keine Nelke dabei, keine Hyazinthe usw. Konkrete Klassen haben instanzierte Objekte, abstrakte Klassen nicht 2P	9
2.	a) Welche Funktion erfüllt der Standard-Konstruktor? b) Woran erkannt man die Konstruktoren einer Klasse? c) Woran erkennt man den Destruktor einer Klasse? d) Wieviele Konstruktoren kann eine Klasse haben? e) Wieviele Destruktoren kann eine Klasse haben?	a) Er wird immer dann verwendet, wenn bei der Generierung (Instanzierung/Definition) von Objekten keine Parameter für den Konstruktor angegeben werden (können). 2P b) Es sind diejenigen Mitgliedsfunktionen, die denselben Namen haben wie die Klasse. 1P c) Es ist diejenige Mitgliedsfunktion, die denselben Namen wie die Klasse trägt, aber mit einem vorangestellten ~-Zeichen. 1P d) Beliebig viele (müssen sich in der Signatur unterscheiden) 1P e) Einen 1P (für beide gilt: wenn keiner definiert ist, macht der Compiler einen)	9

	f) Folgender Klassenrumpf und Konstruktor sind vorgegeben:
	
	`class cC1 {`
	`  char * cp;  // Zeiger auf nullterminierte Zeichenkette`
	
	`public:`
	`  cC1 (char * str_in) {   // Konstruktor`
	
	`  cp = new char [strlen(str_in) + 1];`
	
	`  if (cp)`
	`  strcpy (cp, str_in);`
	`  }`
	`};`
	
	Ergänzen Sie den Destruktor .
		f)
		
		`~cC1 () {`
		`  if (cp)`
		`  delete [] cp;`
		`}`
		
		3P, je 1 Punkt für:
		• Destruktor-Definition
		• Pointer-Abfrage
		• delete-Aufruf mit [] – Angabe beim Destruktor	

3.	Multiple Choice Aufgaben, es können keine, eine oder mehrere Aussagen der Auswahlen richtig sein. Kreuzen Sie nur die richtigen Aussagen an. a) Wegner-Stufen: • Werden Objekte zu Klassen abstrahiert, dann ist das die Stufe 4 nach Wegner. • Der Begriff „Kapsel“ beschreibt den Polymorphismus. • Die Einführung von Klassen beschreibt den Übergang vom Objekt-basierten zum Klassen-basierten Vorgehen. • Durch die Vererbung aus Wegner Stufe 3 ist die Umsetzung des polymorphen Verhaltens möglich. • Durch die Abstraktion aus Wegner Stufe 2 ist die Instanziierung von Objekten möglich. • Keine der Aussagen trifft zu. b) Beziehungen von Objekten: • Durch Ereignisklassen können wir N-zu-M-Beziehungen in 1-zu-1-Beziehungen umwandeln. • Durch die Integration der Kardinalitäten können wir 1-zu-M-	Bewertung: jeweils 3 Punkte für einen korrekten MC-Sechserblock; bei 1 Fehler 2 P, bei 2 Fehler 1 P, bei >= 3 Fehler 0 P (Alternativ: Für jede richtige Teilfrage 0,5 Punkte) *a) F – F – R –R – R – F* *b) F -- F -- F -- R -- R -- F*	12

	Beziehungen in konditional komplexe Beziehungen umwandeln. • Unter der Kardinalität versteht man die vertikale Position der Klasse in der Klassenhierarchie. • Die Kardinalzahlen geben an, mit wie vielen Objekten anderer Klassen Beziehungen bestehen. • Bei einer konditional einfachen Beziehung hat ein Objekt keine Beziehung zu einem anderen Objekt oder eine Beziehung zu genau einem Objekt. • Keine der Aussagen trifft zu. c) Zugriffsrechte: • In C++ gibt es die Zugriffsrechte public, protected, private für Attribute und Methoden. • Methoden mit dem Zugriffsrecht „private“ sind Bestandteil der Schnittstelle einer Klasse. • Wird „private“ abgeleitet, so werden die vererbten public- und protected-Mitglieder in der abgeleiteten Klasse private. • Der this-Zeiger kann verwendet werden um auf die privaten Attribute des eigenen Objekts zuzugreifen. • Die Freund-Funktion einer abgeleiteten Klasse hat keinen Zugriff auf private Mitglieder der Basisklasse. • Keine der Aussagen trifft zu. d) Referenzen: • Eine Referenz ist ein Alias-Name auf ein Objekt und bleibt ständig mit ihm verbunden. • Die Einführung von Referenzen verdoppelt den Laufzeitaufwand für Methodenaufrufe. • Die Einführung von Referenzen reduziert den Laufzeitaufwand für Methodenaufrufe. • Gibt eine Funktion eine Referenz auf ein Objekt zurück, dann muss dieses Objekt über das Verlassen der Funktion hinaus existieren. • Werden Attributswerte in einer Referenz auf ein Objekt verändert, so wird damit auch das Objekt verändert. • Keine der Aussagen trifft zu.	*c) R -- F -- R – R -- R – F* *d) R -- F -- R -- R -- R -- F*	
4.	a) Das folgende Programm enthält syntaktische und logische Fehler.	a) Fehler (max. 7 P)	10

Kennzeichnen Sie mindestens 7 Fehler und beschreiben Sie diese jeweils mit wenigen Worten:

```
#include <iostream>
#include <math.h>
using workspace std;

class complexT {
private:
      double re, im;
      double dist () {
            returning (sqrt(re*re + im*im)));
      }

public:
      complexT (double re_in = 0.0, double im_in = 0.0)
      {re = re_in; im == im_in;}

      void write (void) {
            cout << "Reel: " << re << " Imaginaer: " << im
<< endl;
      }

      operator double (void) { return (int)(dist()+0.5); }

}

int main [] {
      complexT c1(3.3, -4.4), c2(5.5);
      double x, y;

      x = c1
      y = c3;

      cout << "Ergebnis fuer c1: " << x << endl;
      cout <<  Ergebnis fuer c2: " << y << endl;

      return 0;
}
```

b) Beschreiben Sie die Funktion des Programms (ignorieren Sie dabei die Fehler):

```
#include <iostream>
#include <math.h>
using namespace std;


class complexT {
private:
      double re, im;
      double dist () {
            returning (sqrt(re*re + im*im));
      }

public:
      complexT (double re_in = 0.0, double im_in = 0.0)    {re
= re_in; im == im_in;}

      void write (void) {
            cout << "Reel: " << re << " Imaginaer: " << im <<
endl;
      }

      operator double (void) { return (int)(dist()+0.5); }

}

int main [] {
      complexT c1(3.3, -4.4), c2(5.5);
      double x, y;

      x = c1;
      y = c3;

      cout << "Ergebnis fuer c1: " << x << endl;
      cout << _Ergebnis fuer c2: " << y << endl;

      return 0;
}
```

b) Beschreibung: Typtransformierung (cast) (2 P) aus complexT zu double. Es ist der Betrag der komplexen Zahl, also der geometrische Abstand (1 P) zum Nullpunkt (als Absolutwert), gerundet.

5.	<u>Programmieraufgabe am Rechner. Verwenden Sie MS Windows und das MS Visual Studio. Beachten Sie die **Anleitung für die Programmieraufgabe.**</u> Für die Verwaltung von Kinositzplätzen wird ein Programm gebraucht. Es gibt zwei Arten von Kinoplätzen (Parkett und Loge). Ein Kinoplatz kann belegt oder frei sein. Implementieren Sie dazu ein sicheres Array aus einer Klasse cKinoPlatz wie folgt: • Erstellen Sie eine **Klasse cKinoPlatz** mit den Attributen: - art (Loge oder Parkett) - status (frei / belegt) • Erstellen Sie einen Konstruktor, der die ersten 10 Plätze als Loge und den Rest als Parkett initialisiert. (Tipp: Zähler der Erzeugung als statische Variable des Konstruktors bauen.) Der Konstruktor soll jeden dritten Platz als belegt initialisieren, alle anderen Plätze als frei. (Tipp: Modulo-Operation) • Erstellen Sie einen Ausgabeoperator für cKinoPlatz, der die Art und den Status des Kinoplatzes ausgibt. • Implementieren Sie für das sichere Array eine **Klasse cClassArr**, die ein Array aus Objekten der Klasse cKinoPlatz verwaltet. • Erstellen Sie einen Konstruktor, der das Array mit dem new Operator instanziiert. • Erstellen Sie einen Destruktor, der das Array wieder auflöst. • Erstellen Sie einen Subskript-Operator, der eine Referenz auf das indizierte Array-Element vom Typ cKinoPlatz zurückgibt und Fehlindizierungen im Array verhindert. Erkennt der Operator eine Fehlindizierung, gibt er eine Fehlermeldung aus und beendet das Programm. Er muss die belegten Ressourcen dabei freigeben.
```
// Musterloesung Aufgabe 5 Klausur WS 2019/2020
// Reservierungssystem Kinositzplatze als sicheres Array aus Klassen

#include <stdbool.h>      // nur bei Linux
#include <stdlib.h> // nur bei Linux

#include <iostream>
using namespace std;

class cKinoPlatz {
      int art;     // 1 = Parkett, 2 = Loge
      bool status; // frei/belegt
public:
      cKinoPlatz () {
            static int zaehler = 0;

            art = ++zaehler <= 10? 2 : 1;    // die ersten 10
Plaetze sind Loge
            status = zaehler%3? true : false;      // jeder dritte
Platz ist belegt
      }

      friend ostream& operator << (ostream& o, const cKinoPlatz& kp)
{
            o << (kp.art == 2? "Loge, " : "Parkett, ") << (kp.status
== true? "frei" : "belegt");
            return o;
      }
};

class cClassArr
{
      class cKinoPlatz* p;                    // Zeiger auf den
Speicherbereich fuer das Array
      int anz;                          // Anzahl der Array-Elemente
public:
      cClassArr(int);
      class cKinoPlatz& operator [] (const int&);
      ~cClassArr();
};

cClassArr::cClassArr(int anz_in) {
      anz = anz_in < 0 ? 1 : anz_in;          // negative Werte
abfangen

      if (!(p = new class cKinoPlatz[anz])) {// Speicher holen
fehlerhaft ?
```
 | 60 |

Verwenden Sie zum Test Ihrer Klassen folgendes Hauptprogramm:

```
int main() {
	cClassArr Kino(30);		// Kino mit 30 Plaetzen

	for (int i=0; i<33; i++) {
		cout << "Platz Nummer " << i+1 << ": " << Kino[i] << endl;
	}

	return 0;
}
```

Bewertungsschema:	Punktzahl
- Programm lässt sich ohne Fehler/Warnungen übersetzen	6 P
- Codeformatierung	3 P
- Kopfkommentar und sinnvolle Kommentare	3 P
- Artefaktetrennung (z.B. 5 Dateien)	5 P
- Klassendefinition / Konstruktor cKinoPlatz	12 P
- Ausgabeoperator cKinoPlatz	4 P
- Klasse cClassArr / Konstruktor / Destruktor	10 P
- Subsktiptionsoperator Klasse cKlassArr fuer cKinoPlatz	12 P
- Hauptprogramm / Array-Definition / Ausgabeschleife	5 P
Summe:	60 P

Handskizzen können bei der Punktevergabe berücksichtigt werden

---

Platz für Ihre Handskizzen / Notizen:

```
		anz = 0;
	}
}

class cKinoPlatz& cClassArr::operator [] (const int& index) {
	if (index >= 0 && index < anz) {
		return p[index];
	}

	cerr << " ! Fehlindizierung: " << index << " (zulaessig: 0 bis " << anz-1 << ")" << endl;

	if (p) {			// wenn Ressourcen belegt wurden
		delete[] p;		// sauber aufraeumen
		p = 0;
		anz = 0;
	}

	exit(1);
}

cClassArr::~cClassArr() {
	if (p) {			// wenn Ressourcen belegt wurden
		delete[] p;		// sauber aufraeumen
		p = 0;
		anz = 0;
	}
}

int main() {
	cClassArr Kino(30);		// Kino mit 30 Plaetzen

	for (int i=0; i<33; i++) {
		cout << "Platz Nummer " << i+1 << ": " << Kino[i] << endl;
	}

	return 0;
}
```