Bachelorprüfung in "Objektorientierte Programmierung" (OOP) am Studiengang ITS

Allgemeines

Die **Prüfungsfragen** der Bachelorabschlussprüfung **im Fach OOP** sind in folgende **vier Gruppen** eingeteilt:

- OOP-Grundlagen (GL)
- OOP-Mechanismen (ME)
- OOP-Strategien (ST)
- OOP-Advanced (AD)

Die **Prüfungsfragen** bestehen **aus zwei Teilen**, dem "Allgemeinen Teil der Prüfungsfrage" und dem "Spezifischen Teil der Prüfungsfrage", s. Tabelle "Exemplarische Fragestellungen zur Prüfung in OOP" unten. Die Gruppen-Zuordnung finden Sie in der ersten Spalte der Tabelle, den "Allgemeinen Teil der Prüfungsfrage" in der zweiten Spalte, den "Spezifischen Teil der Prüfungsfrage" in der dritten Spalte.

Bitte beachten Sie, dass in der zweiten und dritten Spalte jeweils **nur "Musterbeispiele"** zu Ihrer Orientierung angegeben sind, die tatsächlichen Detailangaben des Allgemeinen und des Spezifischen Teils der Prüfungsfrage am Tag Ihrer Prüfung werden **in abgewandelter Form** sein, s. „Themenliste der LVA OOP“ unten.

Der "**Allgemeiner Teil der Prüfungsfrage**" sollte innerhalb von 10 Minuten der halbstündigen Vorbereitungszeit handschriftlich vorbereitet werden. Der "**Spezifische Teil der Prüfungsfrage**" besteht immer aus einem Implementierungsbeispiel in der Programmiersprache C++, für dessen Programmierung auf einem Rechner des Departments in der Vorbereitungszeit etwa 20 Minuten eingeplant werden sollen.

Sie speichern Ihren **Sourcecode** als Textdatei(en) und einen **Screenshot** Ihres Programm-Outputs auf einem USB-Stick des Departments ab und bringen diesen Stick sowie Ihre handschriftlichen **Notizen** zum Allgemeinen Teil der Prüfungsfrage in den Prüfungsraum mit.

Exemplarische Fragestellungen zur Prüfung in OOP

ID	Allgemeiner Teil der Prüfungsfrage	Spezifischer Teil der Prüfungsfrage
GL-1	Erklären Sie das Prinzip der Kapselung inklusive der dazugehörigen Zugriffsattribute sowie die mit der Kapselung verbundenen Überladungsproblematik in C++.	*Musterbeispiel*: Entwickeln Sie zwei Klassen ('hund' und 'katze'), die beide das Interface 'Tier' implementieren. Der Benutzer bzw. die Benutzerin soll einen Tier-Typ angeben können, anhand dessen ein entsprechendes Objekt instanziiert und anschließend die Methode 'leben()' aufgerufen wird. Diese Methode soll abhängig von der instanziierten Klasse einen individuellen Text ausgeben. Versehen Sie das Interface darüber hinaus mit einer Abfrage einer Integervariablen 'groesse', welche den Status 'private' haben soll..
GL-2	Was sind die Benefits des Überladens von Funktionen in C++, welche Problematiken sind damit verbunden.	*Musterbeispiel*: Es ist eine Klasse für ein Objekt zu definieren, welches ein Rechteck repräsentiert. Das Rechteck besitzt die üblichen Eigenschaften wie Position und Größe. Zum Setzen der Rechteckdaten werden zwei Memberfunktionen SetData(...) definiert.

ID	Allgemeiner Teil der Prüfungsfrage	Spezifischer Teil der Prüfungsfrage
		Die erste Memberfunktion erhält die Rechteckdaten als short-Werte übergeben, während die zweite Memberfunktion die Eigenschaften eines als Parameter übergebenen weiteren Rechtecks verwendet, d.h. es wird eine Kopie des übergebenen Rechtecks erstellt. Die Ausgabe der Rechteckdaten erfolgt mithilfe der Memberfunktion PrintRect(...). In main() werden zwei Rechteck-Objekte definiert, die anschließend mit unterschiedlichen Daten initialisiert werden. Zur Kontrolle werden die Daten der beiden Objekte ausgegeben. Danach werden die Eigenschaften des ersten Objekts dem zweiten Objekt 'zugewiesen' und die Eigenschaften des zweiten Objekts nochmals ausgegeben.
GL-3	Was ist Mehrfachvererbung und wie kann man dabei Namenskonflikte vermeiden?	*Musterbeispiel*: Erstellen Sie eine Klasse Fahrzeug und eine Klasse Kran. Beide Klassen enthalten eine Methode starten(). Die Klasse Fahrzeug erhält zusätzlich eine Methode fahren() und die Eigenschaft MaximalGeschwindigkeit. Die Klasse Kran enthält zusätzlich die Methode heben() und die Eigenschaft Maximalgewicht. Es ist die Mehrfachvererbung anzuwenden, so dass eine Klasse Kranfahrzeug entsteht. Zeigen Sie wie auf die einzelnen Funktionen zugegriffen werden kann, insbesondere bei der Funktion starten(), welche bei beiden Basisklassen vorhanden ist. Zeigen Sie auch, wie die Klasse Kranfahrzeug als Fahrzeug verwendet werden kann?
GL-4	Die Möglichkeiten des Mechanismus ‚Kapselung' in C++ in Hinblick auf Sicherheitsaspekte.	*Musterbeispiel*: Implementieren Sie eine C++-Konsolenapplikation, in der eine Mitarbeiterverwaltung in einem Unternehmen realisiert wird. Es soll zu jedem Mitarbeiter eine Personal-Nummer, ein Gehalt, Geburtsdatum, Bankverbindung und der Name verfügbar sein. Durch Kapselung ist ein Sicherheitsstandard sicherzustellen, der eine direkte Manipulation der Daten verhindert.
GL-5	Wozu gibt es abstrakte Klassen? Was sind die Kennzeichen einer abstrakten Basisklasse. Was muss sichergestellt werden, wenn von einer abstrakten Klasse abgeleitet wird?	*Musterbeispiel*: Erstellen Sie eine abstrakte Basisklasse BaseFigur mit den Eigenschaften length und height und der virtuellen Methode calcArea. Leiten Sie nun die Klasse Rectangle und RightAngleTriangle ab und überschreiben Sie die vererbte abstrakte Methode mit der jeweiligen Flächenberechnung.
GL-6	Welche Vor- und/oder Nachteile hat die Übergabe von Parametern an eine Funktion per Pointer bzw. per Referenz? Welche Besonderheiten weisen Referenzen in C++ auf?	*Musterbeispiel*: Schreiben Sie eine Funktion print, die durch einen vector<int> iteriert und jedes Element über cout ausgibt, sowie eine Funktion modify, die jedes zweite Element eines vector<int> nullsetzt und die Anzahl der nullgesetzten Elemente über einen out/by-reference-Parameter zurückliefert. Implementieren Sie jeweils eine Variante mit Pointervariablen und eine eine mit Referenzvariablen. Rufen Sie die Funktionen beispielhaft auf.
ME-1	Erläuterns Sie Nutzen und Möglichkeiten des Exception-Handlings in C++.	*Musterbeispiel*: Implementieren Sie eine Konsolenanwendung, welche eine Eingabe validiert. Handelt es sich um eine Zahl zwischen 1 und 10, so

ID	Allgemeiner Teil der Prüfungsfrage	Spezifischer Teil der Prüfungsfrage
		wird die Eingabe akzeptiert und 1 dazu addiert. Handelt es sich um keine Zahl, so wird eine Exception geworfen (NOT A NUMBER EXCEPTION). Handelt es sich um eine Zahl außerhalb von 1 und 10, so wird eine andere Exception geworfen (NOT IN RANGE EXCEPTION).
ME-2	Erklären Sie den Mechanismus der Dynamischen Allokation und die Operatoren "new" und "delete" zur Umsetzung in C++ sowie die zugehörigen Möglichkeiten der Fehlerbehandlung.	*Musterbeispiel*: Implementieren Sie eine C++-Konsolenanwendung, die zur Laufzeit Speicher für ein Objekt der Klasse "SimpleClass" und ein Array von "SimpleClass"-Objekten allokiert und wieder korrekt freigibt.
ME-3	Die Möglichkeiten und Grenzen des Mechanismus ‚Polymorphismus' bei C++.	*Musterbeispiel*: Zeigen Sie anhand der Klassen Animal, Cat und Dog den Mechanismus "Polymorphismus", implementieren Sie dazu die jeweilige Klasse in C++ und fügen Sie mindestens eine Methode hinzu, um die Animals ‚sprechen' zu lassen, beispielsweise durch die Konsolenausgabe "Miau" bzw. "Wau".
ME-4	Geben sie einen Überblick über Funktionalitäten der STL und legen Sie deren Bedeutung für die industrielle SW-Entwicklung mit C++ dar.	*Musterbeispiel*: Implementieren Sie folgendes Beispiel in C++ unter Verwendung eines STL-Containers Ihrer Wahl: Es gibt einen Verein mit mehreren Mitgliedern. Die Mitglieder werden in einem STL-Container "Verein" (z.B. mit einer STL-Liste) gespeichert. Die Containerklasse "Verein" bietet der Klasse "Mitglieder" die Methoden beitreten(), austreten(), zaehleMitglieder() usw. an.
ME-5	Welchen Zweck erfüllt ein Copy-Konstruktor? Unter welchen Umständen wird ein Copy-Konstruktor automatisch generiert?	*Musterbeispiel*: Implementieren Sie eine Klasse Student mit den Feldern Matrikelnummer, Name und Kurse. Das Feld Kurse ist dabei eine Liste (oder ein beliebiger anderer Container) vom Typ Kurs*, wobei die Klasse Kurs nicht explizit implementiert werden muss. Erstellen Sie einen Copy-Konstruktor für die Klasse Student und demonstrieren Sie dessen Funktionsweise anhand von Beispieldaten.
ME-6	Erläutern Sie, inwieweit der const-Modifizierer die Eigenschaften einer Variablen zur Kompilier- bzw. zur Laufzeit ändert. Welche Vorteile ergeben sich durch die konsistente Verwendung des Modifizierers?	*Musterbeispiel*: Implementieren Sie vier Varianten der Funktion void printMeasurements(float *array, const size_t N), die alle die Elemente des Arrays auf cerr ausgeben. Die Varianten sollen sich ausschließlich in der Anwesenheit bzw. Abwesenheit von const-Modifizierern des array-Parameters (und des zu Grunde liegenden Datentyps) unterscheiden. Rufen Sie alle Varianten beispielhaft mit einer großen Menge (N >> 1 Mio.) von Beispieldaten auf.
ST-1	Erklären Sie die Möglichkeiten der "generischen Programmierung" in C++.	*Musterbeispiel*: Implementieren Sie in C++ mittels Templates eine Funktion "print()" die ein Argument benötigt und dessen Wert auf dem Bildschirm ausgibt. Definieren Sie weiters eine Array-Struktur "array_t" deren Werte von beliebigem (Template-)Typ sein können und 'N' Elemente beinhaltet. Stellen Sie sicher, dass die Funktion "print()" auch diese Struktur verarbeiten kann.

ID	Allgemeiner Teil der Prüfungsfrage	Spezifischer Teil der Prüfungsfrage
ST-2	Was ist der allgemeine Zweck von Entwurfsmustern und was ist der Vorteil der Verwendung des Singleton-Entwurfsmusters? Bei welchen Anforderungen ist eine Anwendung sinnvoll?	*Musterbeispiel*: Implementieren Sie ein Singleton Entwurfsmuster in C++, welches ein Objekt einer Klasse myEcard instanziiert. Stellen Sie sicher und zeigen Sie in einer Konsolenanwendung, dass durch nochmaligen Aufruf des Konstruktors das bereits erstellte Objekt zurückgegeben wird.
ST-3	Was ist der Vorteil der Verwendung von Interfaces in der Programmierung, wie wirken sich diese auf die Erweiterbarkeit von Programmen aus.	*Musterbeispiel*: Implementieren Sie ein Interface "textable" in C++ das zwei Funktionen bereitstellt: "void writeText(string text)" und "string readText()". Diese Funktionen sollen zum Schreiben und Lesen von Texinhalten dienen. Erstellen sie zwei Klassen "Notizblock" und "TextDatei", die jeweils das Interface textable implementieren. Zeigen Sie anhand der Nutzung dieser Klassen die Vorteile der Kommunikation über Interfaces.
ST-4	Was sind die zentralen Eigenschaften und Fähigkeiten von „smart pointers“ in C++? Welche Grundtypen sind verfügbar und welche Entwicklungsstrategie ist mit ihnen einzuhalten?	*Musterbeispiel*: Implementieren Sie in C++ ein Array von Objekten des Datentyps "Tier" unter Verwendung von "smart pointers" und zeigen Sie die automatische Objektfreigabe durch eine entsprechende Konsolenausgabe.
ST-5	Beschreiben Sie mindestens fünf Strategien zur Fehlervermeidung bei der Programmierung mit C++.	*Musterbeispiel*: Instanziieren Sie eine Eltern- und zwei Kindklassen (kanonische Form) und allozieren Sie dynamisch zwei weiter Objekte der Kindklasse und zeigen Sie dabei die Wirksamkeit der (mindestens) fünf Fehlervermeidungsstrategien.
ST-6	Was ist Mehrfachvererbung? In welchen Szenarien kann Mehrfachvererbung nutzbringend eingesetzt werden?	*Musterbeispiel*: Instanziieren Sie zwei Eltern und eine von diesen erbende Kindklasse und demonstrieren Sie dabei die Auftretenden positiven und negativen Effekte von Mehrfachvererbung.
AD-1	Erläutern Sie Möglichkeiten des Überladens von Operatoren in C++ und diskutieren Sie, welche Probleme dabei auftreten können.	*Musterbeispiel*: Zeigen Sie anhand der Klasse "Bruch" die Überladung des Ausgabeoperators (<<) und eines beliebigen mathematischen Operators.
AD-2	Geben Sie einen Überblick über den Aufbau und die Struktur einer abstrakten Fabrikmethode (Abstract Factory).	*Musterbeispiel*: In einem Zoo wird zwischen Bodentieren (class Soilanimal) und Vögeln (class Bird) unterschieden. Davon wurden bereits vier Tiere abgeleitet: class Lion, class Tiger, class Pecker, class Vulture. Implementieren Sie in C++ eine abstrakte Fabrikmethode und die dazugehörigen abgeleiteten Fabrikmethoden, mit deren Hilfe die entsprechenden Bodentiere oder Vögel instanziiert werden können. Zeigen Sie das Beispiel anhand eines kurzen Aufrufs in der main-Routine.
AD-3	Welche Mechanismen zur „Runtime Type Identification“ (RTTI) werden in C++ geboten?	*Musterbeispiel*: Implementieren Sie in C++ eine Klassenhierarchie mit Basisklasse "Fahrzeug", für die Sie drei beliebige Objekte instantiieren. Verwenden Sie daraufhin mindestens zwei unterschiedliche Arten der Laufzeittypidentifikation, um damit eine Konsolenausgabe der Typinformation durchzuführen.

ID	Allgemeiner Teil der Prüfungsfrage	Spezifischer Teil der Prüfungsfrage
AD-4	Was ermöglicht das "Überladen von Operatoren"? Wo wird es üblicher Weise eingesetzt?	*Musterbeispiel*: Implementieren Sie in C++ die Addition von zwei komplexen Zahlen unter Zuhilfenahme des Überladens von Operatoren.
AD-5	Was bedeutet das Schlüsselwort auto? Ist es zur Kompilier- oder zur Laufzeit relevant? Welche Vorteile bietet es?	*Musterbeispiel*: Schreiben Sie eine Funktion in C++, die alle Elemente eines vector<pair<string, string>> auf cout ausgibt. Implementieren Sie zwei Varianten der Ausgabefunktion - eine, die das Schlüsselwort auto nicht verwendet und eine, die das Schlüsselwort auto überall verwendet, wo es möglich und sinnvoll ist. Rufen Sie die Funktion mit Beispieldaten auf und verwenden Sie typedef-Deklarationen zur besseren Lesbarkeit.
AD-6	Wofür kann das nullptr-Literal in C++ verwendet werden? Welche Art von Aufrufen vereinfacht es?	*Musterbeispiel*: Implementieren Sie zwei Funktionen bool CheckIfNull(const void * const) und bool CheckIfNull(const int * const). Rufen Sie beide Funktionen mit nullptr als Argument sowie mit einer passenden Alternative auf, die zum gleichen Ergebnis führt.

Tabelle Exemplarische Fragestellungen zur Prüfung in OOP

Themenliste der LVA OOP

1. OO-Überblick und OO-Konzepte
2. Strukturen und Klassen
3. Zugriffsattribute
4. Konstruktor und Destruktor, Copy-Konstruktor
5. Dynamische Speicherverwaltung
6. Vererbung und Mehrfachvererbung
7. Abstrakte Klassen
8. Virtuelle Elementfunktionen
9. Überladen von Operatoren
10. RTTI
11. Templates
12. Exceptions
13. STL
14. Neuerungen durch C++20/23
15. Effektives C++ und Fehlervermeidungsstrategien