ulm university universität
uulm

Labor Softwareentwurf mit Multiparadigmen-Programmiersprachen
H. Falk, A. Luppold

Versuch 3
vom 10. November 2014

Bearbeitung bis: 30. November 2014

Objektorientierte Programmierung I

Versuch 3-1: Theorie (15 Punkte)

a) Was ist eine virtuelle Methode?

b) Was versteht man unter einer Memberfunktion? Wie unterscheiden sich diese von Funktionen außerhalb einer Klasse? Wie ändert sich das Verhalten, wenn eine Memberfunktion `static` deklariert wird?

c) Was ist ein Destruktor? Wieso ist das in C++ verwendete Konstruktor-/Destruktor-Konzept bei Java nicht sinnvoll?

d) Welche Formen von Vererbung kennt C++? Umreißen Sie knapp die Unterschiede.

e) Was versteht man unter dem Begriff „const-correctness“?

f) Wieso können nicht alle Operatoren innerhalb einer Klasse überladen werden?

Versuch 3-2: Erste Schritte (15 Punkte)

a) Erstellen Sie für Ihre selbsterstellte Liste (**nicht** die `std::list`-Version) aus dem vorherigen Versuch eine eigene Klasse. Die Klasse soll wie in Versuch 2-2 als eigene Bibliothek umgesetzt werden. Achten Sie darauf, dass sich Ihre Klasse innerhalb eines eigenen Namespaces befindet.

b) Erstellen Sie ein kleines Testprogramm für Ihre neue Klasse.

c) Vergleichen Sie Ihre Implementierung in Versuch 2 mit Ihrer jetzigen Implementierung als Klasse. Wie unterscheiden sich die Ansätze, wenn Sie mehrere Listen gleichzeitig benötigen? Erkennen Sie noch weitere Vor- oder Nachteile zwischen den beiden Ansätzen?

Versuch 3-3: Abstrakte Klassen (20 Punkte)

a) Laden Sie sich die bereitgestellte Datei `arr_interface.hpp` herunter. Implementieren Sie anhand dieser die abstrakte Klasse ArrOp, die von der Klasse ArrInterface erbt. Implementieren Sie die Methoden add(), sub(), mul(), div() und assign().

b) Erläutern Sie, wieso Sie in diesem konkreten Fall Funktionen als virtual bzw. nicht als virtual deklariert haben.

Versuch 3-4: Interfaces (30 Punkte)

a) Implementieren sie die Klassen ArrStorageVector und ArrStoragePointer, die von ArrOp erben und das vorgegebene Interface ArrInterface implementieren.
ArrStorageVector soll intern einen std::vector verwenden.
ArrStoragePointer soll Speicher über new allokieren und auf Zeigern operieren. Achten sie darauf, dass beim Löschen von Elementen zuvor allokierter Speicher wieder freigegeben wird und keine Memory Leaks entstehen. Stellen Sie weiter sicher, dass sich Instanzen der Klasse problemlos kopieren lassen. (Hinweis: Erstellen Sie hierzu einen copy constructor).

b) Welche Auswirkungen hat das vorgegebene Interface auf die Performanz?

c) Warum benutzt das Interface einen std::shared_ptr?

d) Warum ist im Interface ein leerer Destruktor definiert?

e) Wieso gibt es im Interface zwei getElement()-Methoden?

Versuch 3-5: Überladen von Operatoren (20 Punkte)

a) Überladen Sie für ArrInterface den << Operator so, dass der Vektor mittels C++ Streams ausgegeben werden kann.

b) Überladen sie folgende Operatoren: +=, *=, −=, /=, +, −, *, /. Es soll hierbei möglich sein beliebige von ArrInterface abgeleitete Klassen mit den Operatoren zu verwenden.

c) Schreiben Sie ein Testprogramm, das die überladenen Operatoren der vorherigen Teilaufgaben nutzt.

d) Welche Vor- und gegebenenfalls Nachteile erkennen Sie zwischen dem Überladen der Operatoren, und dem Verwenden von Memberfunktionen, die eine vergleichbare Funktionalität bereitstellen (add(), mult,...).