Beispielangaben zu
OOP (Objekt-Orientierte Programmierung)

SS 2009

Beispiel 5

Relevante Themen:

- Generizität
- Iteratoren
- Algorithmen
- C++0x
    - Variadic Templates
    - Rvalue reference
    - static_assert
- RTTI

Design und Implementierung:

Im letzten optionalen Beispiel wollen wir den Stoff des Skriptums verlassen und uns dem kommenden C++ Standard zuwenden. In dieser Angabe wird der derzeitige Status N2857 des neuen Standards C++0x genannt[1, 2].

Das Ziel des kommenden Standards ist es, Alternativen zu derzeit unsicheren Techniken, wesentliche Erweiterungen in der Standardlibrary und verbesserte Performance zu schaffen, dabei aber einfacher zu lernen zu sein. Der letzte Punkt ist deshalb sehr wichtig, weil selbst erfahrene C++ Programmierer immer Anfänger bei neuen Bibliotheken oder neuen Programmier-Paradigmen sind.

In diesem Beispiel gibt es drei Unteraufgaben. Diese können jeweils unabhängig von einander gelöst und mit der Datei verwendung.cpp überprüft werden. Um zwischen der eigenen Implementierung und der Implementierung der STL wechseln zu können gibt es drei Makros: SOLVED_1, SOLVED_2 und SOLVED_3. Es ist darauf zu achten, dass die eigenen Klassen und Algorithmen im Namensbereich `Ti` implementiert werden.

1.) Shared Pointer (20 Punkte)

`auto_ptr` löst ein sehr spezifisches Problem: Es wird sichergestellt, dass ein Speicher der im Client mit new angefordert wurde auch wieder freigegeben wird. Er hat allerdings aufgrund seiner Kopiersemantik ein Problem: Er kann nicht in Containern verwendet werden.

Eine Alternative zu `auto_ptr` in C++0x ist `shared_ptr`. Auch sie können statt normalen Zeigern eingesetzt werden, da sie syntaktisch gleich zu verwenden sind. Und auch sie haben die Eigenschaft, dass wenn das Objekt nicht mehr referenziert wird, dieses automatisch freigegeben wird. `shared_ptr` hat aber den Vorteil, dass problemlos Kopien mit dem Kopierkonstruktor und Zuweisungsoperator angelegt werden können.

Durch Referenzzählung muss sichergestellt werden, dass die Ressource automatisch freigegeben wird. Dazu muss bei jeder Zuweisung und jedem Kopiervorgang der Referenzzähler erhöht und bei jedem `reset` der Referenzzähler dekrementiert werden.

Das Interface ist in shared_ptr.h angegeben (siehe auch Seite 628 in [2]). Das Klassen-Template `shared_ptr` kann über eine new-Expression initialisiert werden. Zusätzlich zu dem im C++0x definierten `shared_ptr` Interface ist eine `make_shared` Funktion zu erstellen, welche eine neues Objekt in einem `shared_ptr` eingekapselt. Da der Konstruktor ja potentiell beliebig viele Parameter haben kann, müssen *Variadic Templates* [2] dafür verwendet werden.

Tipp: Die Implementierung ist wesentlich einfacher wenn der Referenzcounter mit einem zweiten `new` erzeugt wird. Das ist laut C++0x auch erlaubt, obwohl eine einzelne Allokation aus nahe liegenden Gründen natürlich zu bevorzugen wäre.

Letztendlich ist noch die Funktion `shared_dynamic_cast` zu erstellen, welche es ermöglichen soll einen dynamischen Cast zwischen `shared_ptr` durchzuführen.

Implementieren Sie dieses angegebene Interface im Namensbereich Ti vollständig und definieren sie dann das Makro SOLVED_1.

2.) Array Container (10 Punkte)

Implementieren Sie ein Array fixer Größe im Namensbereich Ti wie in array.hpp angegeben (siehe auch Seite 790 in [2]).

Es ist natürlich auch ein (trivialer) random-access Iterator von begin() und end() zurückzugeben.

Auf Seite 100 im Skriptum wird darauf eingegangen wie benutzerdefinierte Typen die gleiche Syntax wie eingebaute Typen erhalten können. Im derzeit gültigen Standard gibt es allerdings keine Möglichkeit, folgendes zu erreichen:
array<int,3> a = {1,2,3};

Genau das ist jetzt mit Initializer lists möglich. Diese Funktionalität ist leider noch nicht im auf dem an den Laborrechnern installierten GCC vorhanden. Implementieren Sie deshalb nur ein Funktions-Template array<T>&& make_array() welches nur ein leeres Array zurückgibt.

Der Operator &&, welcher in g++43 schon implementiert ist, ermöglicht dabei Verschiebesemantik. Es ist damit möglich Werte die in einer Funktion am Stack liegen mittels `std::move` zurückzugeben.

Stellen Sie schließlich mit `static_assert` sicher, dass das Array nicht mit 0 Elementen erzeugt werden kann. Die im Standard beschriebenen Vorkehrungen für diese Situation können demnach ignoriert werden.

Haben Sie diesen Teil abgeschlossen, dann definieren Sie das Makro SOLVED_2.

3.) Algorithmen (10 Punkte)

Algorithmen können in C++ generisch und direkt ausgedrückt werden. Dies geschieht zu einem durch Verwendung eines Funktions-Templates, wodurch der Algorithmus für unterschiedliche Typen verwendet werden kann. Zusätzlich kann mittels Iteratoren der Algorithmus von einem spezifischen Container entkoppelt werden.

Implementieren Sie den Algorithmus `mean_mark` mit dem Interface wie in mean_mark.hpp mittels Iteratoren. Dieser soll die Durchschnittsnote über alle Typen (die die Methode `mark()` haben) zurückgeben.

In dem Algorithmus `mean_mark_student` geben Sie die Durchschnittsnote von Studenten zurück. Stellen Sie hier mit RTTI zur Laufzeit sicher, dass nur von Untertypen von `Student` eine Note berücksichtigt wird.

Der letzte Algorithmus `remove_greater` ist wieder für beliebige Objekte und Container. Mit dieser Funktion sollen alle Objekte gelöscht werden, deren Note größer als die übergebene Note ist (für Rückgabewert siehe im Skriptum Seite 183).

Definieren Sie das Makro SOLVED_3 wenn sie mit diesem Teil der Aufgabe fertig sind.

Anforderungen:

Benutzen Sie zur Programmentwicklung auch ein Makefile, welches zumindest die Targets „clean“, „all“ und „run“ enthält. Mittels „make run“ soll das Programm direkt aufgerufen werden können.

Es sind die Programmierrichtlinien der LVA einzuhalten.

Beschreiben Sie mit Post-Conditions wie sich das Array und der Shared Pointer nach Aufruf der Methode sich verhalten. C++0x gibt das bereits für die nicht beispiel-spezifischen Methoden vor.

Diese Übung ist optional, es können aber **keine** Punkte vergeben werden, wenn

- keine Abgabe erfolgt.
- die Abgabe mit dem auf den Rechnern installierten g++43 mit den Optionen `-std=c++0x -pedantic` nicht fehlerfrei kompiliert.
- valgrind für dieses Beispielprogramm anzeigt, dass der Speicher nicht vollständig freigegeben wird (natürlich ohne ein `delete` hinzuzufügen).
- Keines der SOLVED Makros aktiviert ist

Achtung:

Die Datei verwendung.cpp ist **unverändert** zu verwenden!
Der Compiler muss statt mit g++ mit g++43 aufgerufen werden.
Im Gegensatz zu den Beispielen bis jetzt ist die Verwendung der STL und Boost untersagt!

Abgabe:

Legen Sie ein Verzeichnis „Beispiel5“ an, in welches Sie alle Ihre Programmdateien hinein geben.
Packen Sie das Verzeichnis folgendermaßen:

tar –zcvf Beispiel5.tgz Beispiel5

und geben Sie dieses Archivfile elektronisch ab

Es ist kein Protokoll notwendig.

Bei der Abgabe wird überprüft ob die Beispiele selbst gelöst wurden und vollständig verstanden werden.

Links:

[1] Wikipedia contributors. General information about C++0x.
retrieved 02.06.2009, updated 21 May 2009
http://en.wikipedia.org/wiki/C++0x

[2] ISO/IEC. Working Draft, Standard for Programming Language C++.
updated 23.03.2009, version N2857=09-0047
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2857.pdf