Übungen zur Vorlesung

"Objektorientiertes Programmieren im Wissenschaftlichen Rechnen"

Dr. O. Ippisch, Dr. C. Engwer

Abgabe am 20. 07. 2010 in der Vorlesung

Dieses Blatt ist das letzte Blatt der Vorlesung. Die hier gestellte Aufgabe zählt nicht mehr zur Bestehensgrenze, es ist allerdings möglich sich mit den hier erreichten Punkten zu verbessern.

ÜBUNG 1 TYPE ERASURE (BONUSAUFGABE)

Generische Programmierung mit Templates bietet viele Möglichkeiten flexiblen, hoch effizienten Code zu schreiben. Programme mit dynamischem Polymorphismus hingegen sind in der Regel einfacher zu entwickeln und zu warten. Sie bieten ausserdem eine höhere Flexibilität, da sie Austauschbarkeit zur Laufzeit erlauben.

In der Praxis, besteht ein Programm aus performance kritischen Bereichen und solchen, wo die Kosten von dynamischem Polymorphismus nicht ins Gewicht fallen. Daher hätte man gerne ein Hilfsmittel, um in bestimmten Bereichen, oder ab einem gewissen Abstraktionslevel den statischen Polymorphismus zu verstecken und den Nutzer hier von unnötigen Templatekonstrukten zu bewahren.

Die Methode des `type erasure` bietet hierzu die entsprechenden Möglichkeiten. Beispiele für Bibliotheken, die dieses Verfahren verwenden sind `boost::any` oder `adobe::any_iterator`.

Beispiel:

```
#include <list>
#include <boost/any.hpp>

struct Foo {};

int main() {
    // some arbitrary data
    int i;
    double d;
    Foo f;

    // store any data
    std::list<boost::any> many;

    many.push_back(boost::any(i));
    many.push_back(boost::any(d));
    many.push_back(boost::any(f));

    return 0;
}
```

Wir wollen jetzt versuchen eine Konstruktion selbst zu programmieren. Das Ziel ist es einen abstrakten Stack zu haben, der eine Menge von `int` speichert:

```
class Stack {
    void push_back(int); // append an element
    void pop_back();     // delete last element
    int back();          // read last element
};
```

Dieser abstrakte Stack soll von einem beliebigen Container, der selbst diese 3 Methoden zur Verfügung stellt initialisiert werden können (z.B. `std::vector`, `std::deque`, `std::list`).

Der Trick ist nun folgender. Der `Stack` soll intern einen Pointer auf den eigentlichen Container speichern. Der Container-Typ ist aber nicht a-priori bekannt, daher wird ein virtuelles Interface `StackInterface` für den Container eingeführt. Dieses Interface enthält genau die Methoden, die

der Stack hat, aber es sind alles rein virtuelle Methoden. Der `Stack` enthält jetzt einen Pointer auf `StackInterface` und alle Aufrufe von `push_back`, etc. werden an die entsprechende Methode der `StackInterface` Objektes weitergeleitet.

Jetzt kommt das zweite Problem. Man kann dem `StackInterface` keinen Pointer auf z.B. `std::vector` zuweisen. Um dies zu bewerkstelligen, führt man eine eine templatetierte Wrapper Klasse `StackImpl` ein, die das virtuelle Interface erfüllt. Der Template Parameter entspricht jetzt dem Typen der eigentlichen Implementierung (z.B. `std::vector`), die Klasse speichert eine Referenz, oder einen Pointer auf ein Objekt dieser Implementierung und leitet alle virtuellen Funktionaufrufe statisch an dieses Objekt weiter.

Hinweis: Alternativ kann zur Implementierung von `StackImpl` *auch das Curiously Recuring Template Pattern verwendet werden.*

Man könnte jetzt relativ einfach die Klassen so programmieren, dass man folgenden Aufruf verwenden kann:

```
std::vector v;
Stack s = StackImpl(v);
```

Um die ganze Konstruktion mit `StackInterface` und `StackImpl` vom Nutzer zu verstecken, erhält zu guter Letzt der `Stack` noch einen templatisierten Konstruktor, dieser bekommt einen beliebigen Container `T`, der das Interface statisch erfüllt. In diesem Konstruktor wird nun ein dynamischer Wrapper `StackImpl<T>` auf dem Heap angelegt und dessen Pointer als `StackInterface*` gespeichert.

1. Schreiben Sie ein Testprogramm, welches überprüpft, dass Ihre `type erasure` Konstruktion funktioniert.
2. Implementieren Sie `type erasure` für das eben beschriebene Beispiel der `Stack` Klasse.

12 Punkte

Wie immer gilt: Kommentieren Sie Ihr Programm. Erklären Sie was Sie tuen.