Dr. M. Grabert
Johannes Mayer

Objektorientierte Programmierung mit C++ (WS 2001/2002)

Abgabetermin: 13. Dezember 2001

7 Sichere Vektoren (3 Punkte)

Bei der Klasse `vector<T>` fällt ein Zugriff außerhalb der Array-Grenzen gar nicht auf! Dies muss sich ändern und mit Vererbung ist das ja auch kein Problem – hoffen wir mal. Also schreiben Sie eine Klasse `svector<T>`, die von `vector<T>` erbt und den Operator `[]` neu implementiert, so dass bei Zugriff außerhalb der Array-Grenzen eine `out_of_range`-Exception geworfen wird. Ansonsten soll diese Klasse dieselben Methoden wie `vector<T>` besitzen.

8 Wer führt das Protokoll? (4 Punkte)

Viele Programme machen irgendwelche Ausgaben in Log-Files. Dies können wir natürlich schön kapseln durch die Klasse `log`, die alles in eine Datei schreibt. Soweit der *Stand*. Jetzt kommt Ihnen noch die Idee, dass die Logs auch (zusätzlich) auf die Standardausgabe geschrieben werden sollen. Also bilden Sie eine Unterklasse von `log`, die dies übernimmt. Natürlich könnte man auch zählen, wieviele Logs ausgegeben wurden und diese über die Methode `size()` verfügbar machen. Also schreiben Sie von der abgeleiteten Klasse wiederum eine Unterklasse, die `size()` implementiert. Aber nicht genug. Ihr Chef hat gehört, dass man Vererbung mittels „Delegation“ simulieren kann, d. h. Sie erzeugen eine neue Klasse (ohne Vererbung) und geben Ihr ein Objekt ihrer „Oberklasse“. Aufrufe von „ererbten“ Methoden reicht diese Klasse einfach an das „Oberklassen“-Objekt weiter. Schreiben Sie also – damit sich Ihr Chef freut – noch die beiden Klassen mittels Delegation (und ohne Vererbung).

Beispiel zu Delegation:

```
class A {
public:
    A() {}
    void method();
}
class B : class A {
public:
```

```
    B() : A() {}
}
```

Dann besitzt die Klasse `B` die von `A` ererbte Methode `method()`. Man kann `B` aber auch mittels Delegation realisieren.

```
class B_delegate {
    A a;
public:
    B_delegate() : a() {}
    void method() { a.method(); }
}
```

Dabei muss man natürlich „ererbte“ Methoden definieren. Jedoch bestehen Sie nur aus einem Aufruf der entsprechenden Methode des `A`-Objektes `a`.

9 Wir brauchen einen Verwalter! (3 Punkte)

Sie bekommen eine (minimale) Klasse *student* ähnlich wie die Klasse `student` aus der Vorlesung. Jeder Student wird eindeutig über sein Login identifiziert. Außerdem stehen bei jedem Studenten seine C++-Punkte. Alles Unwichtige haben wir weggelassen :-)). Nun sollen Sie die Studenten, die erzeugt und wieder zerstört werden, verwalten. Wie im `main`-Teil des Beispielprogrammes zu sehen ist, sollen Sie zumindest drei Methoden implementieren, mit denen der Punktedurchschnitt aller Studenten berechnet werden kann, alle Studenten (d. h. alle momentan existierenden Studenten-Objekte) ausgegeben werden können und ein Studenten-Objekt zu einem gegebenen Login gesucht werden kann. Wie man in `main` sieht, werden die angelegten Studenten nicht explizit in den Verwalter aufgenommen oder aus ihm wieder entfernt. Dies soll im Konstruktor bzw. Destruktor passieren.

Viel Spass und Erfolg!