Objektorientierte Programmierung mit C++ (WS 2010)

Dr. Andreas F. Borchert, Tobias Brosch
Institut für Angewandte Informationsverarbeitung
Universität Ulm

Blatt 14: Abgabetermin 09. Februar 2011 11 Uhr

Wiederholung

Geht die Inhalte der Vorlesung nochmals durch. Sind Euch noch Dinge unklar, schreibt bitte rechtzeitig Eurem Übungsleiter eine E-Mail mit den entsprechenden Fragen.

Rekursive Templates

Praxisbeispiel

Angenommen, Wir möchten eine mehrdimensionale Matrixklasse schreiben. Die Anzahl der Dimensionen soll zur Compiletime feststehen. Wie können Wir jedoch einen Eintrag oder auch die Größe des Arrays möglichst leserlich angeben? Eine komfortable Notation sähe beispielsweise so aus:

```
Mat<4> mat(10, 3, 4, 2); // 10x3x4x2-dimensional matrix
entry = mat[10, 3, 2, 1]; // entry at (10, 3, 2, 1)
```

Dies verlangt jedoch nach einem Konstruktor, sowie einem `operator[]`, der je nach Anzahl der Dimensionen verschieden viele Argumente übergeben bekommt. Es ist möglich Funktionen zu schreiben, die beliebig viele Argumente übergeben bekommen (z.B. printf). Allerdings verlieren Wir damit eine Überprüfung des Aufrufs durch den Compiler, sodass Fehler in der Benutzung erst zur Runtime auftreten.

Eine andere Möglichkeit das Indizierungsproblem zu lösen und dennoch einen gut lesbaren Quellcode zu erhalten, ist Folgende (vergleiche `boost::multi_array`):

```
Mat<4> mat(extents[10][3][4][2]); // 10x3x4x2-dimensional matrix
entry = mat[extents[10][3][2][1] ]; // entry at (10, 3, 2, 1)
```

In dieser Aufgabe werden Wir sehen, wie sich dies mit Hilfe eines rekursiven Templates leicht realisieren lässt.

Aufgabe

Eure Aufgabe ist es, die Syntax `extents[10][3][4][2]` mit Hilfe einer Template-Klasse zu ermöglichen. Es ist insbesondere nicht notwendig, eine mehrdimensionale Matrixklasse zu schreiben.

Generelle Idee

Durch wiederholten Aufruf des Operators `[]`, werden Objekte mit einem um Eins höheren Template-Parameter `N` vom Typ `std::size_t` erzeugt, die intern die Einzeldimensionen in einem statischen Array speichern.

Tipps zur Implementierung

- Schreibt eine Template-Klasse `Extents`, welche als Template-Parameter eine Variable `N` vom Typ `std::size_t` übergeben bekommt (`std::size_t` verhält sich wie ein `unsigned int`).
- Intern soll ein statisches Array der Größe `N` die einzelnen Dimensionen speichern.
- Es gibt einen privaten Konstruktor (Wir möchten nicht, dass außer uns jemand ein Objekt erzeugt), der ein Array der Länge `N` übergeben bekommt und in sein internes Array kopiert (Tipp: `std::copy`).
- Da der `operator[]` der Klasse mit dem Template-Parameter `N-1` ein Objekt mit dem Parameter `N` erzeugen können soll, muss die Klasse `Extents<N-1>` als `friend` deklariert werden.
- Um etwas Ausgabe zu erhalten, bietet es sich an auch einen Ausgabeoperator zu schreiben. Damit automatisch der Template-Parameter abgeleitet werden kann, ist es gut diesen außerhalb der Klasse zu definieren. Damit dieser dennoch auf das interne Array zugreifen kann, sollte die Methode als `friend` gekennzeichnet werden.
- Das Kernstück Unserer Klasse ist der `operator[]`, der eine Variable `n` vom Typ `std::size_t` übergeben bekommt und ein neues `Extents<N+1>`-Object zurückliefert (sollte dazu natürlich die Dimensionen von sich selbst plus die neue übergebene Dimension der Größe `n` dem Konstruktor des `Extents<N+1>`-Objektes in einem zuvor generierten Array übergeben).

Benötigen Wir nur noch eine Möglichkeit an ein Objekt der Klasse `Extents<N>` heranzukommen. Dazu spezialisieren Wir unsere Template-Klasse `Extents` im Fall `N=0`. Der Konstruktor soll privat sein (Tipp: Es ist hilfreich hier einfach nur den Default-Konstruktor zu implementieren, der nichts macht). Die Methode `operator[]` sollte natürlich eine Dimension übergeben bekommen und ein `Extents<1>`-Objekt zurückliefern. Und um an ein Objekt von `Extents<0>` heranzukommen definieren wir eine konstante Klassenvariable `extents` vom Typ `Extents<0>`, die natürlich öffentlich ist.

Testet Euer Konstrukt kurz in der `main`-Methode Eures Programms.

Submission

Einreichen der Lösung mit:

```
submit cpp 14 main.cpp
```

Viel Spaß!