Objektorientierte Programmierung mit C++ (WS 2010)

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

Blatt 6: Abgabetermin 01. Dezember 2010

1 Operatoren und Friends

Im Gegensatz zu Java, lassen sich in C++ Operatoren selbst definieren. Damit können wir bei unserer Matrixklasse aus dem letzten Übungsblatt beispielsweise statt `M1.add(M2)` einfacher schreiben `M1 += M2`.

Wie geht das? Ein Operator ist eine Methode, deren Namen mit `operator` beginnt und anschließend die entsprechenden Operatorsymbole beinhaltet. Für den Klammmeroperator `[]` z.B. `operator[]` (die Methode also z.B. `double operator[](int i)`) oder für den Operator + z.B. `Mat2d operator+(const Mat2d& M2) const`. Eine ganz gute Übersicht über die verschiedenen Operatoren und wie diese als Memberfunktion oder außerhalb einer Klasse definiert werden können, findet ihr hier: http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B.

Bei manchen Operatoren kommt es vor, dass sich diese nicht innerhalb einer Klasse definieren lassen (siehe untenstehende Aufgaben). Dennoch will man in der Implementierung eines Operators vielleicht auf private Membervariablen zugreifen können. Daher gibt es in C++ das Konzept der `friend`s. Möchtet Ihr beispielsweise in der Methode `Mat2d operator+(double d, const Mat2d& M)` auf private Variablen der Klasse `Mat2d` zugreifen, so könnt Ihr diese als `friend` in die Klasse aufnehmen und an anderer Stelle implementieren: In der Klasse schreibt Ihr dann `friend Mat2d operator+(double d, const Mat2d& M);` und an der Stelle, an welcher die Methode implementiert wird, `Mat2d operator+(double d, const Mat2d& M) { /* Implementation */ }` (an dieser Stelle also keine `friend` Deklaration mehr notwendig!).

2 Matrix- und Vektor-Klassen (extended)

Ergänzt die beiden Klassen um folgende Operatoren (ergänzt dazu die Dateien `Mat2d.h Mat2d.C Vec.h Vec.C MVoperators.C` in `bspB06.tgz` auf unserer Webseite. Das Meiste ist dort bereits implementiert, die fehlenden Stellen sind mit TODO gekennzeichnet):

- `operator(int i, int j)`, der Gleiches leistet, wie die Methoden `getElem` bisher (das sind pro Klasse 2 Operatoren).
- Operatoren für die Addition, Subtraktion, Multiplikation (wir betrachten **keine** Ma-

trixmultiplikation, sondern elementweise Multiplikation) und Division. Dabei wollen wir natürlich auch Varianten wie zum Beispiel `3 + M` (also einem `double` eine Matrix zu addieren und wieder eine Matrix zu erhalten), sowie die Varianten mit gleichzeitiger Zuweisung (z.B. `+=`). Das sind pro Klasse jeweils 20 Methoden, wobei nicht Alle davon als Memberfunktionen definiert werden können (welche sind das und warum geht dies nicht?).

- Den Multiplikationsoperator, der uns eine Matrix mit einem Vektor multipliziert und einen Vektor zurückliefert.

Es ist dabei empfehlenswert, sich im voraus folgende Gedanken zu machen:

- Wie sehen die Operatormethoden aus?
- Insbesondere: Wann wird eine Referenz zurückgegeben, wann ein Objekt?
- Welche Bestandteile können `const` deklariert werden (die Methoden selbst, als auch die Variablen)?
- Können manche Operatoren mit Hilfe anderer Operatoren implementiert werden? (hilft vlt. „`+=`“ bei „`+`“?)

Schreibt die Methoden, die Ihr außerhalb einer Klasse definiert in eine Datei namens `MVoperators.h` und deren Implementierung in `MVoperators.C`.

3 Tests

Gute Software wird harten Tests unterzogen. Da wir natürlich fast ausschließlich gute Software schreiben, müssen wir unsere Operatoren auch testen. Folgende Matrizen und Vektoren seien von der Größe $m \times n$ bzw. n. Mit $M1 = 1$, ist also eine Matrix gefüllt mit Einsen gemeint. Implementiert hierfür noch die benötigten zusätzliche Operatoren (wir sind ja jetzt geübt). Die Dateien `MVopsTests.h` und `MVopsTests.C` beinhalten folgende Tests, die anschließend (mit Euren zusätzlichen Operatoren) kompilieren und laufen sollten):

```
M1 = 1;
M2 = 0;
M2 += M1 * 2; // M2 = 2
M2 /= 4; // M2 = .5
M2 *= 2; // M2 = 1
M2 -= 0; // M2 = 1
-1 + M2 + 1 == 1 - M2 - 1 + 2; // ?
4 * M2 * .25 == 4 / M2 / 4; // ?
M1 * M2 == M1 / M2; //?
M3 = rand(); // every element random! use operator(i, j)
M3 * M3 == M3^2; // ? // elementwise test with operator(i, j)
// same tests for vectors V1 and V2
// ...
//
```

```
V = 1;
M1 * V == n; //?
```

Teilt die Deklaration und Implementierung der Tests in die Dateien `MVopsTests.[hC]` auf. Wer übrigens noch nicht so recht überzeugt ist, dass dies deutlich schöner ist als in Java, sei dazu ermutigt, obige Tests in Java-Notation aufzuschreiben mit Methoden namens `add, mul, div, ....`

Einreichen der Lösung mit:

```
submit cpp 6 Mat2d.h Mat2d.C Vec.h Vec.C MVoperators.C
```

Beachtet dabei, dass Ihr lediglich diese fünf Dateien verändert und diese mit den restlichen, unveränderten Dateien, kompilieren und die Tests erfolgreich durchlaufen. Lasst bitte die TODO-Markierungen stehen!

Viel Spaß!