Dirk Praetorius,
Marcus Page,
Thomas Führer

Wintersemester 2012/13
11.12.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 11

Aufgabe 11.1*. Erweitern Sie die Klasse `Bruch` aus Aufgabe 10.1–10.2 um einen Konstruktor mit der Signatur `Bruch(long z, unsigned long n)`. Überladen Sie auch die Operatoren `+,-,*,/`, welche die Addition, Subtraktion, Multiplikation und Division zweier Brüche bewerkstelligen. Das Ergebnis soll wiederum ein Bruch sein, so dass eine Syntax wie `C = A*B` möglich ist. Achten Sie auf etwaige Sicherheitsabfragen und überlegen Sie wie man zwei Brüche addiert, multipliziert etc. Speichern Sie den Source-Code unter `bruch.{hpp,cpp}` in das Verzeichnis `serie11`.
Hinweis: Nähere Informationen zum Überladen von Operatoren in C++ finden Sie beispielsweise auf `www.c-plusplus.de/forum/232010-full`

Aufgabe 11.2*. Schreiben Sie eine Klasse `RealVektor` welche Vektoren $x \in \mathbb{R}^n$ verwaltet. Die Klasse enthalte `double * entries` für die Einträge und `int n` für die Länge. Implementieren Sie den Konstruktor `RealVektor()`, welcher `n=0` und `entries=NULL` setzt und den Konstruktor `RealVektor(int n)`, welcher Speicher für ein Array der Länge n anlegt und die Wert mit 0 initialisiert. Schreiben Sie auch den Destruktor welcher den Speicher wieder freigibt. Eine `get` Funktion die die Länge n zurückgibt wird auch nützlich sein. Implementieren Sie dann noch die Methode `resize(int nnew)` welche den Vektor auf die Länge `nnew` kürzt oder verlängert. Gegebenfalls müssen neue Einträge wieder mit 0 initialisert werden. Was können Vor- bzw. Nachteile einer manuellen Speicherverwaltung sein? Speichern Sie den Source-Code unter `RealVektor.{hpp,cpp}` in das Verzeichnis `serie11`.

Aufgabe 11.3*. Erweitern Sie die Klasse aus Aufgabe 11.2 folgendermaßen: Überladen Sie den `[]`-Operator um auf das j-te Element zugreifen zu können (Achtung: Indizierung soll bei 0 starten). Überladen Sie dann auch noch die Operatoren `+,-,*,/`, welche das elementweise Addieren, Subtrahieren, usw. von Vektoren gleicher Länge realisieren sollen. Achten Sie auf etwaige Sicherheitsabfragen. Wird z.B. versucht auf ein Element $j >= n$ zuzugreifen, so soll eine Fehlermeldung am Bildschirm ausgegeben werden. Speichern Sie den Source-Code unter `RealVektor.{hpp,cpp}` in das Verzeichnis `serie11`.

Aufgabe 11.4*. Die Frobeniusnorm einer Matrix $A \in \mathbb{R}^{M \times N}$ ist durch

$$\|A\|_F := \Big(\sum_{j=1}^{M} \sum_{k=1}^{N} A_{jk}^2 \Big)^{1/2}$$

definiert. Schreiben Sie eine Funktion `frobeniusnorm`, die für gegebene Matrix A die Frobeniusnorm berechnet und zurückgibt. Dabei soll die Matrix A als `vector< vector<double> >` übergeben werden. Sie können Vektoren $x \in \mathbb{R}^n$ auch als $n \times 1$-Matrizen auffassen. Überladen Sie die Funktion `frobeniusnorm` sodass auch Vektoren vom Typ `RealVektor` (vgl. Aufgabe 11.2) übergeben werden können. Die Frobeniusnorm des Vektors ist dann einfach seine euklidische Länge. Speichern Sie den Source-Code unter `frobeniusnorm.cpp` in das Verzeichnis `serie11`.

Aufgabe 11.5. Erweitern Sie die Klasse `Bruch` aus Aufgabe 11.1 um die Vergleichsoperatoren `<,<=,>,>=`. Implementieren Sie auch die Klassenmethode `operator double() const;` welche einen Typecast von `Bruch` auf `double` realisiert. Der Code

```
double wert = Bruch(1.0,2.0);
cout << "Wert = " << wert << endl;
```

soll also möglich sein und als Ausgabe `Wert = 0.5` liefern. Speichern Sie den Source-Code unter `bruch2.cpp` in das Verzeichnis `serie11`.

Aufgabe 11.6. Machen Sie sich mit dem Unterschied zwischen Pointer und Referenzen vertraut. Geben Sie die Unterschiede zwischen den zwei aus der Vorlesung bekannten `swap`-Funktionen mit eigenen Worten wieder. Wiederholen Sie auch nochmals die Begriffe *Call by Value* und *Call by Reference*.

Aufgabe 11.7. Ein Paar ist ein C++-Datentyp, der zwei Werte möglicherweise unterschiedlichen Typs enthält. Ein Paar von double-Werten wird beispielsweise mit `pair<double,double>` bezeichnet. Ein Paar von Werten wird beispielsweise mithilfe des Aufrufs `pair<double,double>(5.,3.)` erstellt. Um Paare zu verwenden müssen Sie die Header-Datei `map` einbinden! Schreiben Sie eine Funktion `minmax`, die eine Liste von Gleitkommazahlen erhält, das Minimum und Maximum dieser Liste ermittelt und in Form eines Paares zurückgibt! Geben Sie das Minimum und Maximum in der `main`-Funktion aus! Speichern Sie den Source-Code unter `minmax.cpp` in das Verzeichnis `serie11`. *Hinweis:* Auf den ersten Wert eines Paares kann mit `.first` auf den zweiten mit `.second` zugegriffen werden. Beispiel:

```
pair<int, double> x(5, 17.4);
cout << x.first << ", " << x.second << endl; // Ausgabe: 5, 17.4
```

Aufgabe 11.8. Schreiben Sie eine Klasse `Hangman` mit den Methoden `guessChar`, `solve` und `newString`. Die Klasse soll nun einen string der Länge n speichern, den es zu erraten gilt. Nach und nach dürfen mittels `guessChar` Buchstaben geraten werden, wobei die Methode immer den Index, bzw. die Indizes der eingegebenen Buchstaben auf dem Bildschirm ausgibt. Falls der eingegebene Buchstabe im gesuchten Wort nicht vorkommt, soll eine entsprechende Meldung ausgegeben werden. Der Spieler verliert, wenn er 8 mal falsch geraten hat. Wie lässt sich das geschickt umsetzen? Schreiben Sie alle nötigen Zugriffsfunktionen und Konstruktoren und außerdem die Methoden `solve` und `newString` um das Rätsel zu lösen bzw. das Spiel mit einem neuen Wort neu zu beginnen. Testen Sie Ihr Spiel indem Sie mittels einer geeigneten Schleife solange neue Buchstaben raten, bis Sie entweder gewonnen oder verloren haben. Speichern Sie den Source-Code unter `hangman.{hpp,cpp}` in das Verzeichnis `serie11`.