Dirk Praetorius,
Gregor Gantner

Wintersemester 2018/19
24./25.01.2019

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 11

Aufgabe 11.1. Erweitern Sie die Klasse `Fraction` von Folie 229 um

- den Standardkonstruktor (ohne Parameter), der $p = 0$ und $q = 1$ setzt,
- einen Konstruktor, der $p, q \in \mathbb{Z}$ mit $q \neq 0$ als Input übernimmt und den Bruch speichert,
- den Kopierkonstruktor,
- den Zuweisungsoperator und
- den Destruktor.

Stellen Sie mittels `assert` sicher, dass die Übergabeparameter zulässig sind, d.h. $q \neq 0$. Beachten Sie den Fall $q < 0$, bei dem intern $(-p)/|q|$ gespeichert wird. Testen Sie ihre Implementierung geeignet! Speichern Sie den Source-Code unter `fraction.{hpp/cpp}` in das Verzeichnis `serie11`.

Aufgabe 11.2. Implementieren Sie eine Methode `reduce` für die Klasse `Fraction` aus Aufgabe 11.1, um einen Bruch auf die gekürzte Form $x = p/q$ zu bringen (mit $p \in \mathbb{Z}$ und $q \in \mathbb{N}$ teilerfremd). Verwenden Sie einen beliebigen Algorithmus, um den größten gemeinsamen Teiler von Nenner und Teiler zu bestimmen, z.B. den Euklid-Algorithmus (VO Folien 96–98 und Folie 106) oder einen Algorithmus für die Primfaktorzerlegung von Integern. Testen Sie Ihre Implementierung geeignet!

Aufgabe 11.3. Implementieren Sie das Type Casting von `Fraction` aus Aufgabe 11.1 auf `double`. Implementieren Sie auch das Type Casting von double auf `Fraction`. Berücksichtigen Sie nur die ersten 9 Nachkommastellen (*Hinweis:* Übersetzen Sie diesen Anteil in einen Bruch mit dem Nenner 10^9 und kürzen Sie dann). Implementieren Sie ferner das Type Casting von `Fraction` auf `int`, das einen Bruch auf die nächste Ganzzahl rundet. Beträgt der Nachkommaanteil im mathematischen Sinn genau 0.5 so unterscheiden Sie die folgenden beiden Fälle: Ist der Bruch positiv, so runden Sie auf; d.h. aus $3/2$ wird 2. Ist der Bruch negativ, so runden Sie ab; d.h. aus $-3/2$ wird -2 Testen Sie Ihre Implementierung geeignet!

Aufgabe 11.4. Überladen Sie die Operatoren $+$, $-$, $*$ und / um die Summe, die Differenz, das Produkt und den Quotienten zweier Brüche im Format `Fraction` aus Aufgabe 11.1 zu berechnen. Stellen Sie bei / mittels `assert` sicher, dass Sie nicht durch 0 dividieren. Das Ergebnis soll in allen Fällen gekürzte Form haben. Überladen Sie außerdem den (unären) Vorzeichenoperator $-$, der zum Bruch x im Format `Fraction` den Bruch $-x$ liefert, und den `<<`-Operator, um einen Bruch $x := p/q$ im Format `Fraction` in der Form `p/q` ausgeben zu können (siehe Folie 289 für ein Beispiel mit der Klasse `Complex` aus der Vorlesung). Testen Sie Ihre Implementierung geeignet!

Aufgabe 11.5. Schreiben Sie eine Klasse `FractionVector` zur Speicherung von Vektoren in $\mathbb{Q}^n$. Hierbei sollen die Länge n (`int`) sowie der Koeffizientenvektor (`Fraction*`) abgespeichert werden. Implementieren Sie

- den Konstruktor,
- den Kopierkonstruktor,
- den Zuweisungsoperator und
- den Destruktor.

Implementieren Sie darüber hinaus die Zugriffsmethoden

- `setCoefficient`, `getCoefficient` für die Vektoreinträge und
- `getLength` für die Länge.

Testen Sie Ihre Implementierung geeignet! Speichern Sie den Source-Code unter `fractionVector.{hpp/cpp}` in das Verzeichnis `serie11`.

Aufgabe 11.6. Schreiben Sie die Methode `sort` für die Klasse `FractionVector` aus Aufgabe 11.5, die einen Vektor aufsteigend sortiert und mit dem sortierten Koeffizientenvektor überschreibt. Der Vektor

$$x = \left(-\frac{1}{2}, \frac{5}{7}, \frac{1}{3}, \frac{0}{1}, \frac{11}{2}, -\frac{7}{8}\right) \in \mathbb{Q}^6$$

soll beispielweise durch Aufruf von `sort` durch

$$x = \left(-\frac{7}{8}, -\frac{1}{2}, \frac{0}{1}, \frac{1}{3}, \frac{5}{7}, \frac{11}{2}\right)$$

überschrieben werden. Wählen Sie einen beliebigen Sortieralgorithmus und verwenden Sie den Type Cast auf `double` aus Aufgabe 11.3 um zwei Brüche miteinander zu vergleichen. Wie groß ist der Aufwand Ihrer Implementierung? Testen Sie Ihren Code geeignet!

Aufgabe 11.7. Schreiben Sie eine Klasse `IVector` zur Speicherung von Vektoren in $\mathbb{N}^n$. Hierbei sollen die Länge n (`int`) sowie der Koeffizientenvektor (`int*`) abgespeichert werden. Implementieren Sie

- den Konstruktor,
- den Kopierkonstruktor,
- den Zuweisungsoperator und
- den Destruktor.

Implementieren Sie darüber hinaus die Zugriffsmethoden

- `setCoefficient`, `getCoefficient` für die Vektoreinträge und
- `getLength` für die Länge.

Speichern Sie den Source-Code unter `iVector.{hpp/cpp}` in das Verzeichnis `serie11`. Schreiben Sie außerdem eine Funktion `primfaktoren`, die für eine natürliche Zahl $M \in \mathbb{N}$ deren Primfaktoren bestimmt und als Vektor $p \in \mathbb{N}^n$ vom Typ `IVector` zurückgibt. Die Koeffizienten p_j des Vektors $p \in \mathbb{N}^n$ sind also Primzahlen, und es gilt $M = \prod_{j=1}^{n} p_j$. Um eine Liste aller möglichen Primfaktoren zu erhalten, verwende man das Sieb des Eratosthenes aus Aufgabe 5.5. Speichern Sie den Source-Code unter `primfaktoren.cpp` in das Verzeichnis `serie11`. Testen Sie Ihre Implementierung geeignet!

Aufgabe 11.8. Schreiben Sie eine Funktion `kgV(a,b)`, die das kleinste gemeinsame Vielfache zweier natürlicher Zahlen $a, b \in \mathbb{N}$ berechnet. Zur Lösung können Sie entweder mit Aufgabe 11.7 die Primfaktoren beider Zahlen berechnen oder den Zusammenhang $a \cdot b = ggT(a,b) \cdot kgV(a,b)$ berücksichtigen. Um den größten gemeinsamen Teiler zu bestimmen, verwenden Sie z.B. den Algorithmus von Euklid (VO Folien 96-98 und Folie 106). Speichern Sie den Source-Code unter `kgv.cpp` in das Verzeichnis `serie11`. Test Sie Ihre Implementierung geeignet!