Dirk Praetorius,
Michele Ruggeri

Sommersemester 2017
16.05.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. Das *Sieb des Eratosthenes* ist ein Algorithmus zur Bestimmung aller Primzahlen kleiner oder gleich einer vorgegebenen Zahl $n_{max} \in \mathbb{N}$. Der Algorithmus sieht folgendermaßen aus:

- Man legt eine Liste $(2, \dots, n_{max}) \in \mathbb{N}^{n_{max}-1}$ an.
- Man streicht aus der Liste alle Vielfachen der ersten Zahl.
- Wähle, solange es noch höhere Zahlen gibt, die nächsthöhere nicht durchgestrichene Zahl und streiche alle ihre Vielfachen.

Schreiben Sie eine Struktur `Eratosthenes` zur Speicherung des Ergebnisses des Algorithmus. Die Struktur besteht aus der oberen Grenze n_{max} (`int`), der Anzahl der Primzahlen $n \leq n_{max} - 1$ (`int`) und dem Vektor in $\mathbb{N}^n$ der Primzahlen (`int*`). Schreiben Sie ferner eine Funktion `Eratosthenes* = doEratosthenesSieve(int nmax)`, die das Sieb des Eratosthenes realisiert. Die Rückgabe verwende die Struktur `Eratosthenes`. Realisieren Sie das Streichen geeignet und rechenökonomisch. Beachten Sie, dass der Vektor der Primzahlen gekürzt auf minimale Länge werden soll. Testen Sie Ihren Code entsprechend! Speichern Sie den Source-Code, aufgeteilt in Header-Datei `eratosthenes.h` und `eratosthenes.c`, in das Verzeichnis `serie09`.

Aufgabe 9.2. Die Primfaktorzerlegung einer natürlichen Zahl $N \in \mathbb{N}$ ist die Darstellung von N als Produkt von Primzahlen, die dann als Primfaktoren bezeichnet werden; z.B. für $N = 180$ hat man die Zerlegung $120 = 2^2 \cdot 3^2 \cdot 5$. Schreiben Sie eine Struktur `Factorization` zur Speicherung der Primfaktorzerlegung. Die Struktur besteht aus der Anzahl der Primfaktoren n in der Zerlegung (`int`), dem Vektor in $\mathbb{N}^n$ der Primfaktoren (`int*`) und dem Vektor in $\mathbb{N}^n$ der Vielfachheiten (`int*`). Im vorherigen Fall ($N = 180$) würden die Members der Struktur beispielweise die Werte `n=3`, `prime=(2,3,5)` und `multiplicity=(2,2,1)` enthalten. Schreiben Sie eine Funktion

`Factorization* = computePrimeFactorization(int N)`,

die die Primfaktorzerlegung einer natürlichen Zahl bestimmt und zurückgibt. Um eine Liste aller möglichen Primfaktoren zu erhalten, verwende man das Sieb des Eratosthenes aus Aufgabe 9.1. Schreiben Sie ferner eine Funktion

`int = recomposeInteger(Factorization* factorization)`,

die das umgekehrte Vorgehen durchführt. Testen Sie Ihren Code entsprechend! Speichern Sie den Source-Code, aufgeteilt in Header-Datei `primefactorization.h` und `primefactorization.c`, in das Verzeichnis `serie09`.

Aufgabe 9.3. Implementieren Sie zwei Funktionen:

- `int = ggT(int a, int b)`, die den größten gemeinsamen Teiler zwei gegebenen natürlichen Zahlen $a, b \in \mathbb{N}$ berechnet und zurückgibt;
- `int = kgV(int a, int b)`, die das kleinste gemeinsame Vielfache zwei gegebenen natürlichen Zahlen $a, b \in \mathbb{N}$ berechnet und zurückgibt.

GgT und kgV kann man über die Primfaktorzerlegung der gegebenen Zahlen bestimmen. Verwenden Sie daher die Struktur und die Funktionalitäten aus Aufgabe 9.2. Speichern Sie den Source-Code unter `gcdAndLcm.c` in das Verzeichnis `serie09`.

C++ Aufgaben

Aufgabe 9.4. Erstellen Sie eine Klasse `SparKonto` mit den Variablen `kontonummer`, `guthaben` und `zinssatz`. Ferner sollen noch die `get` und `set`-Methoden für die Variablen `zinssatz` und `kontonummer` implementiert werden. Um das Guthaben zu ändern schreiben Sie die Methoden `abheben` und `einzahlen`. Beachten Sie, dass Sie bei einem Sparkonto nicht ins Minus gehen können. Der Zinssatz und die Kontonummer dürfen natürlich auch nicht negativ werden. Schließlich implementiere man noch die Methode `berechneGuthaben`. Testen Sie Ihren Code entsprechend. Speichern Sie den Source-Code unter `sparkonto.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.5. Schreiben Sie eine Klasse `University`. Diese soll neben den Feldern `numStudents`, `city` und `name` die Methoden `graduate` und `newStudent` haben. Wird `graduate` aufgerufen, so verringert sich die Anzahl der Studenten um 1, wohingegen `newStudent` die Anzahl um 1 erhöht. Alle Datenfelder sollen als `private` deklariert sein. Sie müssen sich also zusätzlich `get`- und `set`-Methoden schreiben. Testen Sie Ihren Code entsprechend. Speichern Sie den Source-Code unter `University.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.6. Erstellen Sie eine Klasse `Name` welche zwei String-Variablen `vorname` und `nachname` enthält. Implementieren Sie auch die Zugriffsfunktion `setName`, welche einen String übernimmt, diesen in Vor- und Nachname aufteilt und in die entsprechenden Variablen abspeichert. Beachten Sie auch, dass mehrere Vornamen vorhanden sein können! Schreiben Sie weiters eine Methode `printName` die Vor- und Nachname am Bildschirm ausgibt. Bei mehreren Vornamen soll, beginnend ab dem zweiten Vornamen, jeder weitere Vorname mit dem Anfangsbuchstaben und einem Punkt abgekürzt werden! Beim Namen `Max Maxi Mustermann` soll also `Max M. Mustermann` am Bildschirm erscheinen. Speichern Sie den Source-Code unter `name.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.7. Erweitern Sie die Klasse `Triangle` aus der Vorlesung (Folie 216) mit zwei Methoden:

- die Methode `getPerimeter()`, die den Umfang des Dreieckes berechnet und zurückgibt;
- die Methode `isEquilateral()`, die überprüft, ob ein Dreieck gleichseitig ist.

Testen Sie Ihre Implementierung entsprechend!

Aufgabe 9.8. Implementieren Sie eine Methode `reduce` der Klasse `Fraction` aus der Vorlesung (Folie 228). Dabei sollen der Zähler p und Nenner q durch $p_0 \in \mathbb{Z}$ und $q_0 \in \mathbb{N}$ überschrieben werden, wobei $p = gp_0$ und $q = gq_0$, mit $g \in \mathbb{N}$ maximal. Gehen Sie dazu wie folgt vor:

- Für $p = 0$ gilt $p_0 = 0$ und $q_0 = 1$.
- Für $p \neq 0$ ist g der größte gemeinsame Teiler von $|p|$ und q. Diesen können Sie mit dem Euklid-Algorithmus bestimmen. Für $a, b \in \mathbb{N}$ funktioniert dieser Algorithmus wie folgt:
  (i) Im Fall $a = b$, ist der größte gemeinsame Teiler klar.
  (ii) Anderenfalls garantiere $a < b$ durch Vertauschen und ersetze b durch $b - a$.
  (iii) Wiederhole die beiden Schritte (i)–(ii), bis $a = b$ gilt.

Testen Sie ihre Implementierung geeignet!