Dirk Praetorius,
Gregor Gantner

Sommersemester 2018
29.05.2018

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. Erstellen Sie eine Klasse `Kunde` für einen Kunden bei einer Bank. Diese Klasse soll den Namen des Kunden als `string`, den aktuellen Kontostand als `double` und eine PIN als `int` beinhalten. Implementieren Sie neben geeigneten `get` und `set` Methoden noch folgende Klassenmethoden

- `void printBalance()`
  gibt den aktuellen Kontostand am Bildschirm aus.
- `bool checkPIN()`
  liest eine PIN ein und überprüft, ob diese korrekt ist.
- `void drawMoney()`
  überprüft zuerst die PIN, liest den abzuhebenden Betrag ein und gibt den neuen Kontostand am Bildschirm aus. Das Konto darf hierbei nicht überzogen werden. Geben Sie gegebenenfalls eine Warnung am Bildschirm aus.

Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `kunde.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.2. 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. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `University.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.3. Für die Personalabteilung der Universität ist es sehr mühsam, Studenten immer nur einzeln hinzuzufügen oder aus dem System zu löschen. Überladen Sie die Methoden `graduate` und `newStudent` der Klasse `University` aus Aufgabe 9.2 so, dass die Anzahl der abschließenden bzw. neu hinzukommenden Studenten mit übergeben werden kann. Schreiben Sie außerdem Konstruktoren die Ihre Universität mit sinnvollen Daten befüllen. Wird das Objekt nicht direkt initialisiert, so soll `numStudents = 0`, `city = noWhere` und `name = noName` eingetragen werden. Erweitern Sie die Klasse zusätzlich um eine `plot` -Routine, welche sämtliche Daten von `University` ausgibt. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `University.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.4. Erweitern Sie die Klasse `Triangle` aus der Vorlesung (Folien 213ff) 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 9.5. Schreiben Sie eine Klasse `Stoppuhr` welche zur Messung von Rechenzeiten dienen soll. Die Stoppuhr bestehe dabei aus zwei Methoden. Wird die erste aufgerufen, so soll die Zeitmessung gestartet werden. Wird diese Methode nochmals aufgerufen, wird die Zeitmessung gestoppt. Die zweite Methode dient dazu, die Zeit wieder zurückzusetzen. Schreiben Sie die Methoden `pushButtonStartStop` und `pushButtonReset`. Implementieren Sie weiters eine Methode `print`, welche die verstrichene Zeit im Format `hh:mm:ss.xx` ausgibt (Beträgt die gemessene Zeit also zwei Minuten so soll `00:02:00.00` ausgegeben werden). Sie können diese Stoppuhr nun dazu verwenden Zeitmessungen durchzuführen. Verwenden Sie folgenden Code-Ausschnitt, um Ihre Implementierung zu testen:

```
Stoppuhr S;
double sum = 0.0;
S.pushButtonStartStop();
for(int j=0; j<100*1000*1000; ++j)
  sum += 1./j;
S.pushButtonStartStop();
S.print();
```

Was wird hier berechnet? Speichern Sie den Source-Code unter `stoppuhr.{hpp,cpp}` in das Verzeichnis `serie09`.

Hinweis: Verwenden Sie den Datentyp `clock_t` und die Funktion `clock()` aus der Bibliothek `time.h`. Die verstrichene Rechenzeit zwischen zwei Aufrufen von `clock()` in Sekunden erhalten Sie mittels

```
clock_t t1, t2;
double secs;
t1 = clock();
/* ... do some work ... */
t2 = clock();
secs = (double) (t2-t1) / CLOCKS_PER_SEC;
```

Vermutlich ist es auch sinnvoll, eine Variable `isRunning` vom Typ `bool` einzuführen. Beim Aufrufen der ersten Methode wird diese Variable entweder von `false` auf `true` gesetzt oder umgekehrt.

Bonus: Adaptieren Sie obigen Code-Ausschnitt um $\sum_{j=1}^{N} j^3$ für $N = 10^8$ auf zwei verschiedene Arten zu berechnen: Einmal naiv durch Verwenden der Potenzfunktion `pow(j,3)` aus der Mathematik-Bibliothek, einmal clever ohne Verwendung der Mathematik-Bibliothek. Messen Sie auch jeweils die Rechenzeit! Was beobachten Sie?

Aufgabe 9.6. Schreiben Sie eine Klasse `Matrix` zur Speicherung von quadratischen $n \times n$ `double` Matrizen, in der neben vollbesetzten Matrizen (Typ `'F'`) auch untere (Typ `'L'`) und obere (Typ `'U'`) Dreiecksmatrizen gespeichert werden können. Dabei bezeichnet man Matrizen

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix} \qquad L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

als obere bzw. untere Dreiecksmatrix. Mathematisch formuliert, gilt also $u_{jk} = 0$ für $j > k$ bzw. $\ell_{jk} = 0$ für $j < k$. Eine vollbesetzte Matrix werde im Fortran-Format spaltenweise als dynamischer Vektor der Länge $n \cdot n$ gespeichert. Dreiecksmatrizen sollen in einem Vektor der Länge $\sum_{j=1}^{n} j = n(n+1)/2$ gespeichert werden. Implementieren Sie folgende Funktionalitäten:

- Standardkonstruktor, der eine 0×0 Matrix vom Typ `'F'` anlegt
- Konstruktor, bei dem der Typ und die Dimension mit übergeben werden kann
- Destruktor
- `get` und `set`-Methoden für die Matrix-Einträge, den Typ und die Dimension

Dabei hängen insbesondere die `get` und `set`-Methoden für die Matrixeinträge vom Matrixtyp (Dreiecksmatrix!) ab. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `matrix.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.7. Erweitern Sie die Klasse `Matrix` aus Aufgabe 9.6 um

- eine Methode `scanMatrix(char typ, int n)`, die den Typ, sowie die Matrix $A \in \mathbb{R}^{n \times n}$ dem Typ entsprechend von der Tastatur einliest,

- eine Methode `printMatrix()`, die die Matrix am Bildschirm ausgibt,
- eine Methode `columnsumnorm()`, die die Spaltensummennorm

$$\|A\| = \max_{k=0,\ldots,n-1} \sum_{j=0}^{n-1} |a_{jk}|$$

berechnet und zurückgibt,

- eine Methode `rowsumnorm()`, die die Zeilensummennorm

$$\|A\| = \max_{j=0,\ldots,n-1} \sum_{k=0}^{n-1} |a_{jk}|$$

berechnet und zurückgibt.

Beachten Sie, dass die Methoden bei unteren bzw. oberen Dreiecksmatrizen nur auf Koeffizienten a_{jk} bzw. a_{kj} für $0 \leq k \leq j \leq n-1$ zugreifen können. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `matrix2.{hpp,cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.8. Laut der Vorlesung ist der Zugriff auf Members einer Klasse vom Typ `private` nur über `set`- und `get`-Methoden der Klasse möglich. Wie lautet die Ausgabe des folgenden C++ Programms? Warum ist das möglich? Erkären Sie, warum das schlechter Programmierstil ist.

```
#include <iostream>
using std::cout;
using std::endl;

class Test{

private:
  int N;

public:
  void setN(int N_in) { N = N_in; };
  int getN(){ return N; };
  int* getptrN(){ return &N; };

};

int main(){

  Test A;
  A.setN(5);
  int* ptr = A.getptrN();
  cout << A.getN() << endl;
  *ptr = 10;
  cout << ptr << endl;
  cout << A.getN() << endl;

  return 0;
}
```