Prof. Dr. Carsten Meyer
Fachbereich Informatik & Elektrotechnik
Fachhochschule Kiel

Programmieren in C++ – Sommersemester 2014

Aufgabenblatt 4

Abgabe: Donnerstag, 18.06.2015, 11:30

Aufgabe 1 (Besondere Klassenelemente, Klassenvariablen, Kopierkonstruktor):

a) Die Klasse **`Kunde`** soll folgende Attribute besitzen:

```
char* name;
char* ort;
int* alter;
int anzahlEinkaeufe;
double umsatz;
const int kundenId;
static int zaehlerKunden;
static int anzahlKunden;
static int gesamtAnzahlEinkaeufe;
```

Das Attribut **`anzahlEinkaeufe`** zählt die Anzahl der Einkäufe eines Kunden, das Attribut **`umsatz`** gibt den (Gesamt-) Umsatz des Kunden mit seinen Einkäufen an. Die Konstante **`kundenId`** enthält die Kundennummer; sie wird mit Hilfe der Klassenvariablen **`zaehlerKunden`** automatisch inkrementiert, wenn ein Objekt der Klasse **`Kunde`** angelegt wird. Die Klassenvariable **`anzahlKunden`** zählt die aktuelle Anzahl der Objekte der Klasse **`Kunde`** (d.h. im Gegensatz zur Klassenvariablen **`zaehlerKunden`** wird die Klassenvariable **`anzahlKunden`** verringert, wenn ein Objekt der Klasse **`Kunde`** vernichtet wird). Die Klassenvariable **`gesamtAnzahlEinkaeufe`** schließlich zählt die gesamte Anzahl der Einkäufe aller Kunden.

Verfassen Sie

- einen Konstruktor mit Vorgabeargumenten für **`name`**, **`ort`** und **`alter`**,
- einen Destruktor,
- einen Kopierkonstruktor (überlegen Sie, welche Attribute wie gesetzt werden müssen),
- die Methode **`kaufe(double u)`**, mit welcher **`umsatz`** um **`u`** erhöht und gleichzeitig die Anzahl der Einkäufe inkrementiert wird,
- eine Methode **`getKundenId()`**, mit welcher die **`kundenId`** zurückgegeben werden kann,

- eine Klassenmethode **getGesamtAnzahlEinkaeufe()**, mit der die Anzahl der Einkäufe **gesamtAnzahlEinkaeufe** aller Objekte der Klasse **Kunde** bestimmt werden kann,
- eine **print()**–Methode, mit welcher z.B. folgender Text ausgegeben werden kann:

```
Kunde Dagobert aus Entenhausen (ID = 1, 50 Jahre) hatte 2 Einkaeufe
und 7000 Euro Umsatz
```

Testen Sie die Klasse **Kunde** mit dem nachfolgenden Programm:

```
void test( Kunde k )
{
      cout << "Beginn test" << endl;
      Kunde k_a("Sherlock", "Paris", 66);
      k_a.kaufe(2000);
      k_a.kaufe(250);
      k_a.print();
      cout     <<     "gesamte     Anzahl     Einkaeufe:     "     <<
Kunde::getGesamtAnzahlEinkaeufe() << endl;
      Kunde k_b( k );
      k_b.kaufe(80);
      k_b.print();
      cout     <<     "gesamte     Anzahl     Einkaeufe:     "     <<
Kunde::getGesamtAnzahlEinkaeufe() << endl;
      cout << "Ende test" << endl;
}

int main( int argc, char* argv[] )
{
    Kunde k1("Dagobert", "Entenhausen", 50);
    k1.kaufe(1200);
    k1.kaufe(5800);
    k1.print();
    cout     <<     "gesamte     Anzahl     Einkaeufe:     "     <<
Kunde::getGesamtAnzahlEinkaeufe() << endl;

    Kunde k2( k1 );
    cout << "Id von Kunde k2: " << k2.getKundenId() << endl;
    k2.print();
    cout     <<     "gesamte     Anzahl     Einkaeufe:     "     <<
Kunde::getGesamtAnzahlEinkaeufe() << endl;

    Kunde* k3 = new Kunde("Willi", "Berlin", 60);
    k3->kaufe(20);
    k3->kaufe(70);
    k3->kaufe(50);
    k3->print();
    cout     <<     "gesamte     Anzahl     Einkaeufe:     "     <<
Kunde::getGesamtAnzahlEinkaeufe() << endl;
```

```
    Kunde k4 (*k3);
    k4.print();
    cout     <<     "gesamte     Anzahl     Einkaeufe:     "     <<
Kunde::getGesamtAnzahlEinkaeufe() << endl;

    test( k2 );
    cout << "zurueck in main" << endl;

    const Kunde k5("Goofy", "Entenhausen", 18);
    k5.kaufe(111);
    k5.print();
    cout     <<     "gesamte     Anzahl     Einkaeufe:     "     <<
Kunde::getGesamtAnzahlEinkaeufe() << endl;

    delete k3;
    return 0;
}
```

Welche Zeile(n) des Programms müssen auskommentiert werden, damit das Programm kompiliert?

(nach HS Esslingen)

b) Deklarieren Sie den Kopierkonstruktor privat. Welche Zeilen des Testprogramms müssen jetzt auskommentiert werden, damit das Programm kompiliert?

Aufgabe 2 (Matrix-Klasse, Konstruktoren, Destruktor, Friends, Operatorüberladung):

In dieser Aufgabe soll eine Klasse zur Addition und Multiplikation von dynamischen Matrizen entwickelt werden, wobei der Speicherplatz zur Laufzeit dynamisch reserviert werden soll. Außerdem soll es möglich sein, mit Hilfe des Index-Operators [ ] [ ] wie gewohnt auf die Elemente der Matrix zuzugreifen (Beispiel: **`m[i][j]`** ist das Element in Zeile **`i`** und Spalte **`j`**). Die entsprechende Klasse wird deshalb ein dynamisches Element enthalten, das die Matrix als eindimensionales Feld mit Elementen, die selbst eindimensionale Zeilen sind, adressiert.
Im folgenden ist eine Header-Datei **`matrix.h`** gegeben, in der die Klasse **`Matrix`** mit Hilfe einer Klasse **`Zeile`** deklariert ist. Die Klasse **`Zeile`** ist ein eindimensionales Feld von **`int`**-Werten; weiterhin hat die Klasse einen überladenen Indexoperator **`[]`**. Die Klasse **`Matrix`** besitzt ein dynamisches Element **`mat`**, das ein Feld von Zeigern auf **`Zeile`**-Objekte adressieren soll. **`mat`** ist also ein Zeiger auf Zeiger.

```
// ------------------------------------------------------
// matrix.h:  Darstellung von dynamischen Matrizen
// ------------------------------------------------------

#ifndef _MATRIX_H_
#define _MATRIX_H_

#include <iostream>
#include <cassert>
using namespace std;

class Zeile
{
    int *z;
    int size;

  public:

    // zu implementieren: Konstruktor und Destruktor
    Zeile( int s);
    ~Zeile();

    // Indexoperator [] ist (in zwei Versionen) bereits implementiert für die Klasse Zeile
    int& operator[](int i)
    {
       assert(i >= 0 && i < size);
       return z[i];
    }

    const int& operator[](int i)  const
    {
       assert(i >= 0 && i < size);
       return z[i];
    }
};
```

```
class Matrix
{
  private:
    Zeile **mat;              // Zeiger auf "Zeilen"-Vektor
    int nrows, ncols;         // Zeilen- und Spaltenzahl

  public:

    // zu implementieren: Zwei Konstruktoren, ein Kopierkonstruktor und ein Destruktor
    Matrix( int _nrows , int _ncols);
    Matrix( int z, int s, int wert);
    Matrix( const Matrix& );
    ~Matrix();

    // zu implementieren: Zuweisungsoperator für tiefe Kopie (falls in Vorlesung besprochen)
    Matrix& operator=( const Matrix& ma );

    int  getRows()  const { return nrows; }
    int  getCols()  const { return ncols; }

    // zu implementieren: Indexoperator [] (in zwei Versionen) für die Klasse Matrix
    Zeile& operator[](int i);
    const Zeile& operator[](int i) const;

      // zu implementieren: Transponierte der Matrix
      Matrix transpose();

    // zu implementieren: Ausgabefunktion, i.e. print() oder operator<<
    void print();
//        friend  std::ostream&  operator<<(std::ostream&  os,  const
Matrix& m);

    friend Matrix operator+(const Matrix& ma, const Matrix& mb);
    friend Matrix operator*(const Matrix& ma, const Matrix& mb);
};

// zu implementieren: zwei globale Operatorfunktionen für Addition und Multiplikation
Matrix operator+(const Matrix& ma, const Matrix& mb);
Matrix operator*(const Matrix& ma, const Matrix& mb);
#endif
```

a) Implementieren Sie die fehlenden Methoden und Funktionen:

- Einen Konstruktor und einen Destruktor für die Klasse **`Zeile`**. Der Konstruktor erhält einen Integer-Wert als Parameter (**`int s`**); er soll ein **`int`**–Feld der Länge **`s`** dynamisch anlegen und die Membervariablen **`*z`** und **`size`** entsprechend initialisieren. Der Destruktor soll den Speicherplatz wieder freigeben.

- Der Konstruktor der Klasse **`Matrix`** soll ein Feld mit **`nrows`** Zeigern auf Objekte vom Typ **`Zeile`** anlegen. Anschließend wird der Speicherplatz für die Zeilen selbst in einer Schleife dynamisch reserviert. Der Destruktor gibt im Gegenzug als erstes den Speicherplatz für die einzelnen Zeilen frei. Danach wird der Platz für das Feld von Zeigern auf Zeile **`mat`** selbst freigegeben. Außerdem soll ein überladener Konstruktor

implementiert werden, der als zusätzlichen Parameter eine Variable vom Typ **`int`** erhält, mit der alle Matrixelemente initialisiert werden. Weiterhin soll ein Kopierkonstruktor implementiert werden.

- Überladen Sie den Zuweisungsoperator für die Klasse **`Matrix`** für tiefe Kopie.
  Hinweis: Falls der Zuweisungsoperator für tiefe Kopie noch nicht in der Vorlesung besprochen wurde (was vermutlich der Fall ist), lassen Sie diesen Teil der Aufgabe weg. Achten Sie dann jedoch darauf, daß Sie im Hauptprogramm z.B. bei der Matrixaddition den Kopierkonstruktor verwenden (den Sie für tiefe Kopie überladen haben). Also:
  ```
  Matrix a, b;
  Matrix c = a + b; // Kopierkonstruktor
  ```
  anstelle von
  ```
  Matrix a, b, c;
  c = a + b; // Zuweisungsoperator
  ```

- Der Index-Operator (in einer non-**`const`** und einer **`const`**-Version) für die Klasse **`Matrix`** soll zu einem vorgegebenem Index **`i`** die i-te Zeile liefern (wobei der zulässige Bereich für den Index mit einem „**`assert`**“ verifiziert wird). Die Implementierung der Indexoperatoren soll **`inline`** erfolgen.
  Hinweis: Bei der Auswertung **`m[2][3]`** wird also zunächst der Index-Operator der Klasse **`Matrix`** aufgerufen, der die Zeile zum Index 2 liefert. Danach wird für diese Zeile der Index-Operator der Klasse **`Zeile`** aufgerufen. Er liefert eine Referenz auf den **`int`**-Wert beim Index 3.

- Die Methode **`transpose()`** soll die transponierte Matrix berechnen, wobei ein neues Matrix-Objekt zurückgegeben wird.

- Überladen Sie den Additionsoperator „+“ und den Multiplikationsoperator „*“ für die Klasse **`Matrix`**. Die Addition C=A+B und die Multiplikation C=A*B zweier Matrizen A und B erzeugt jeweils ein neues Matrix-Objekt C.
  Hinweis: Falls die Operatorüberladung noch nicht in der Vorlesung besprochen sein sollte, ersetzen Sie bitte
  `operator+(`…**`)`**
  durch
  `add(`…**`)`**
  und
  `operator*(`…**`)`**
  durch
  `mult(`…**`)`**

- Implementieren Sie die Ausgabefunktion, die die Matrix auf dem Bildschirm ausgibt. Dabei sollen die einzelnen Werte der Matrix pro Zeile durch je drei Leerzeichen getrennt hintereinander ausgegeben werden.
  Hinweis: Falls die Überladung des Ausgabeoperators „<<“ noch nicht in der Vorlesung besprochen sein sollte, ersetzen Sie bitte
  `operator<<(`…**`)`**
  durch eine einfache Ausgabefunktion
  `print(`…**`)`**

(nach Univ. Basel)

b) Testen Sie die Klasse **`Matrix`**, indem Sie eine Matrix 3×4-Matrix A erzeugen und folgendermaßen initialisieren:

Zeile 1:	1	2	3	4
Zeile 2:	11	12	13	14
Zeile 3:	21	22	23	24

Erzeugen Sie eine Matrix B als Kopie von A, addieren Sie beide Matrizen und geben Sie das Ergebnis auf dem Bildschirm aus. Berechnen Sie dann die transponierte Matrix B^t von B, multiplizieren Sie A mit B^t und geben Sie das Ergebnis auf dem Bildschirm aus.

Die Lösung der Aufgabe soll die Implementierung des Testprogramms und die Bildschirmausgabe umfassen.

Aufgabe 4 (Klassentemplates):

Machen Sie aus der **`Matrix`**-Klasse ein Klassentemplate und testen Sie das Klassentemplate für die Typen **`int`** und **`double`** gemäß Aufgabe 3b. Die **`double`**-Matrix gehe dabei aus der **`int`**-Matrix durch Division aller Matrixelemente durch 10.0 hervor.

Die Lösung der Aufgabe soll die Implementierung des Testprogramms und die Bildschirmausgabe umfassen.

Hinweis: Falls Sie Aufgabe 3 nicht bearbeitet haben sollten, definieren Sie bitte das Klassentemplate basierend auf der in Aufgabe 3a angegebenen Headerdatei **`matrix.h`**. Implementieren Sie zusätzlich die Methode

```
Matrix operator+(const Matrix& ma, const Matrix& mb)
```

bzw. (siehe unter Aufgabe 3)

```
Matrix add(const Matrix& ma, const Matrix& mb)
```

und deklarieren Sie im Hauptprogramm zwei Matrizen, die Sie anschließend addieren.