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

Programmieren in C++ – Sommersemester 2015

Aufgabenblatt 5

Abgabe: Mittwoch, 01.07.2015, 11:30

Aufgabe 1 (Klassentemplates, Operatorüberladung, Friend-Funktionen):

Die Klasse **`Vector`** liegt wie folgt vor:

```
// Datei: vector.h

#ifndef _VECTOR_H_
#define _VECTOR_H_

#include <iostream>

using namespace std;

class Vector
{
      int* v;
      int anzahl;

public:
       Vector(int anzahl = 10);
       Vector( const Vector& vec ); // Kopierkonstruktor
       ~Vector();
       friend bool kleiner( const int& a, const int& b );
       int min() const;
       int getAnzahl() const;
       int operator[]( const int i ) const;
       void setVector( int* sv, int sanzahl);
       friend ostream& operator<< ( ostream& os, const Vector& v );
};

#endif
```

```
// Datei: vector.cpp

#include "vector.h"

Vector::Vector( int a ) : anzahl(a)
{
    v = new int[a];
    for ( int i = 0; i < a; i++ )
        v[i] = i;
}

Vector::Vector( const Vector& vec )
{
    …
}

Vector::~Vector()
{
    …
}

bool kleiner( const int& a, const int& b )
{
     return ( a < b );
}

int Vector::min() const
{
     int min = v[0];
     for ( int i = 1; i < anzahl; i++ )
     {
         if ( v[i] < min )
             min = v[i];
     }
     return min;
}

int Vector::getAnzahl() const
{
    …;
}

int Vector::operator[] ( const int i ) const
{
    …;
}

void Vector::setVector( int* sv, int sanzahl )
{
     delete[] v; // alten Inhalt loeschen
     anzahl = sanzahl;
     v = new int[anzahl];
     for ( int i = 0; i < anzahl; i++ )
         v[i] = sv[i];
     return;
}
```

```
ostream& operator<< ( ostream& os, const Vector& v )
{
     for ( int i = 0; i < v.anzahl; i++ )
         os << v[i] << ", ";
     return os;
}
```

a) Ergänzen Sie die fehlenden (mit „…“ markierten) Implementierungen.

b) Machen Sie aus der **`Vector`**-Klasse ein Klassentemplate und testen Sie das Klassentemplate für die Typen **`int`** und **`double`**, gemäß folgendem – ggf. anzupassendem und zu erweiterndem – Schema:

```
int main(int argc, char *argv[])
{
    Vector v1;
    cout << "v1: " << v1 << endl;

    Vector v2(8);
    cout << "v2: " << v2 << endl;
    cout << "Minimum von v2: " << v2.min() << endl;

    Vector v3(v2);
    cout << "v3: " << v3 << endl;
    cout << "Anzahl von v3: " << v3.getAnzahl() << endl;

    if ( kleiner( v3[2], v2[5] ) )
        cout << v3[2] << " ist kleiner als " << v2[5] << endl;

    int arr[5] = { 10, 5, 2, 3, 12 };

    Vector v4;
    cout << "v4: " << v4 << endl;
    v4.setVector( arr, 4 );
    cout << "v4 nach set: " << v4 << endl;
    cout << "Minimum von v4: " << v4.min() << endl;
    cout << "Anzahl von v4: " << v4.getAnzahl() << endl;

    return 0;
}
```

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

c) Die Klasse **Punkt** sei wie folgt deklariert:

```
// Datei: punkt.h

#ifndef _PUNKT_H_
#define _PUNKT_H_

#include <iostream>
#include <cmath>

using namespace std;

class Punkt
{
      float x,y;

public:
       Punkt( float x0 = 0, float y0 = 0 );
         Punkt ( const Punkt& p ); // Kopierkonstruktor
       ~Punkt();
       void setX( float x0 );
       void setY( float y0);
       float getX() const;
       float getY() const;
       float abstand00() const;
       float abstand( const Punkt& p ) const;
        friend ostream& operator<< ( ostream& os, const Punkt& p );
};

inline Punkt::Punkt( float x0, float y0 ) : …
inline Punkt::Punkt( const Punkt& p ) : …
inline Punkt::~Punkt() {}
inline void Punkt::setX( float x0 ) …
inline void Punkt::setY( float y0 ) …
inline float Punkt::getX() const { … }
inline float Punkt::getY() const { … }
inline float Punkt::abstand00() const { return sqrt( x*x + y*y ); }
inline float Punkt::abstand( const Punkt&p ) const
              { return sqrt( (x-p.x)*(x-p.x) + (y-p.y)*(y-p.y) ); }
inline ostream& operator<< ( ostream& os, const Punkt& p )
              { os << "(" << p.x << ", " << p.y << ")"; return os; }

#endif
```

Ergänzen Sie die fehlenden (mit „…“ markierten Implementierungen).
Kann man das Klassentemplate auch mit dem Typ **Punkt** starten? Welche fehlenden Methoden bzw. Funktionen müssen ergänzt werden?

Aufgabe 2 (Vererbung, Komposition, virtuelle Funktion, Polymorphie):

Geometrische Formen in der Ebene (wie allgemeine Formen, Punkte, Kreise, Quadrate, Dreiecke usw.) haben Gemeinsamkeiten und jeweils individuelle Eigenschaften. In C++ kann dies mit dem Konzept der Vererbung („ein Kreis *ist* eine Form“) oder mit dem Konzept der Komposition („ein Dreieck *hat* drei Punkte“) dargestellt werden.

a) In diesem Programm sollen mehrere Klassen definiert werden:

i. Die Basisklasse **`Form`** besitzt zwei **`protected`** Datenkomponenten *x* und *y* vom Typ **`double`** für die Koordinaten. Außerdem gibt es die folgenden **`public`** –Methoden:
   - Einen Konstruktor mit zwei Argumenten vom Typ **`double`** mit den Defaultwerten 0. Die Argumente dienen zur Initialisierung von *x* und *y*.
   - Eine Methode **`verschieben`** mit zwei Argumenten *dx* und *dy* vom Typ **`double`**, die zu den Komponenten *x* bzw. *y* addiert werden. Der Ergebnistyp ist **`void`**.
   - Eine Methode **`ausgeben`** ohne Argumente, die die Komponenten *x* und *y* als Text ausgibt. Der Ergebnistyp ist **`void`**.

ii. Von der Basisklasse **`Form`** soll eine Klasse **`Kreis`** abgeleitet werden. Dabei stellen die Komponenten *x* und *y* die Koordinaten des Mittelpunkts dar. Die Zugriffsrechte auf die Komponenten der Basisklasse sollen unverändert bleiben. Zusätzlich besitzt ein **`Kreis`** die Datenkomponente *r* für den Radius (Datentyp **`double`**) und im **`public`** – Teil die folgende Methoden:
   - Einen Konstruktor mit drei Argumenten vom Typ **`double`** mit den Defaultwerten 0. Die Argumente dienen zur Initialisierung von *x, y* und *r*. Für die Initialisierung von *x* und *y* ist der Konstruktor der Basisklasse aufzurufen.
   - Eine Methode **`aufblaehen`** mit einem Argument *f* vom Typ **`double`** mit Defaultwert 1. Der Radius des Kreises wird mit *f* multipliziert. Der Ergebnistyp ist **`void`**.
   - Eine Methode **`ausgeben`** ohne Argumente, die die Komponenten *x, y* und *r* als Text ausgibt. Der Ergebnistyp ist **`void`**.

iii. Von der Basisklasse **`Form`** soll eine Klasse **`Rechteck`** abgeleitet werden. Dabei stellen die Komponenten *x* und *y* die Koordinaten der linken unteren Ecke dar. Die Zugriffsrechte auf die Komponenten der Basisklasse sollen unverändert bleiben. Zusätzlich besitzt ein **`Rechteck`** die Datenkomponenten *b* und *h* für Breite und Höhe des Rechtecks und im **`public`** –Teil die folgende Methoden:
   - Einen Konstruktor mit vier Argumenten vom Typ **`double`** mit den Defaultwerten 0. Die Argumente dienen zur Initialisierung von *x, y, b* und *h*. Für die Initialisierung von *x* und *y* ist der Konstruktor der Basisklasse aufzurufen.
   - Eine Methode **`dehnen`** mit zwei Argumenten *fx* und *fy* vom Typ **`double`** mit den Defaultwerten 1. Die Breite bzw. Höhe des Rechtecks wird mit *fx* bzw. *fy* multipliziert. Der Ergebnistyp ist **`void`**.
   - Eine Methode **`ausgeben`** ohne Argumente, die die Komponenten *x, y, b* und *h* als Text ausgibt. Der Ergebnistyp ist **`void`**.

iv. Die Klasse **`Dreieck`** wird nicht von der Basisklasse abgeleitet, sondern besitzt drei Komponenten *a, b* und *c* vom Typ **`Form`**. Dabei stellen diese Komponenten die Koordinaten der drei Ecken dar (die Klasse **`Form`** sollte hier besser **`Punkt`** heißen). Zusätzlich besitzt ein **`Dreieck`** im **`public`** –Teil die folgende Methoden:

- Einen Konstruktor mit sechs Argumenten vom Typ **`double`.** Die Argumente dienen zur Initialisierung der drei Ecken *a, b* und *c*. Für die Initialisierung von *a, b* und *c* soll jeweils der Konstruktor der Klasse **`Form`** aufgerufen werden.
- Eine Methode **`verschieben`** mit zwei Argumenten *dx* und *dy* vom Typ **`double`**, die die drei Ecken *a, b* und *c* mit der entsprechenden Methode aus der Klasse **`Form`** verschiebt. Der Ergebnistyp ist **`void`**.
- Eine Methode **`ausgeben`** ohne Argumente, die die Komponenten *a, b* und *c* als Text ausgibt. Der Ergebnistyp ist **`void`**.

v. Schreiben Sie ein Hauptprogramm, in dem die folgenden Aktionen mit Hilfe der definierten Konstruktoren, Methoden usw. ausgeführt werden:

- Konstruieren Sie eine Form *f* (d.h. einen Punkt) mit den Koordinaten (1,2).
- Konstruieren Sie einen Kreis *k* mit dem Mittelpunkt (2,3) und dem Radius 1.
- Konstruieren Sie ein Rechteck *r* mit der linken unteren Ecke (3,2), mit Breite 1 und Höhe 1.
- Konstruieren Sie ein Dreieck *d* mit den Eckpunkten (1,1), (2,3), (3,2).
- Geben Sie alle vier Objekte aus mit der jeweiligen Methode **`ausgeben`**.
- Geben Sie den Kreis *k* und das Rechteck *r* auch mit der Methode **`ausgeben`** aus, die von der Basisklasse **`Form`** geerbt, aber verdeckt wurde.
- Verschieben Sie die Form *f* um (1,1), also um 1 in *x*-Richtung und um 1 in *y*-Richtung.
- Verschieben Sie den Kreis *k* um (2,2).
- Verschieben Sie das Rechteck *r* um (3,3).
- Verschieben Sie das Dreieck *d* um (1,1).
- Blähen Sie den Kreis *k* um den Faktor 1.5 auf.
- Dehnen Sie das Rechteck *r* um den Faktor 2 in *x*-Richtung und um den Faktor 3 in *y*-Richtung.
- Geben Sie nochmal alles aus, wie oben.

vi. Fügen Sie am Ende Ihres Programms eine kleine Ergänzung an: Vereinbaren Sie ein Feld mit drei Komponenten vom Typ „Zeiger auf die Klasse **`Form`**“. Erzeugen Sie mit dem Operator **`new`** drei Instanzen verschiedener Klassen und speichern Sie ihre Adressen in diesem Feld ab. Die Instanzen sollen sein:

- eine Form mit den Koordinaten (2,3),
- ein Kreis mit Mittelpunkt (3,3) und Radius 2,
- ein Rechteck mit linker unterer Ecke (2,2), Höhe 3 und Breite 3.

Geben Sie in einer Schleife alle drei Instanzen aus mit Hilfe der Methode **`ausgeben`**. Beachten Sie, daß alle drei Instanzen nur als Form ausgegeben werden, also der Kreis ohne Radius, das Rechteck ohne Breite und Höhe. Fügen Sie anschließend in das Programm das Wortsymbol **`virtual`** an der richtigen Stelle ein und beachten Sie, daß jetzt die Ausgabe korrekt erfolgt (mit Radius, Höhe und Breite).

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

(Quelle: Gruener, Univ. Karlsruhe)

b) Leiten Sie die Klasse **`Rechteck`** nunmehr **`protected`** statt **`public`** von **`Form`** ab. Welche Zeilen im Hauptprogramm müssen auskommentiert werden, damit das Programm kompiliert und funktioniert? (Die Lösung dieser Aufgabe ist das modifizierte Hauptprogramm **`main.cpp`**).

Aufgabe 3 (Vererbung, virtuelle Funktion, abstrakte Funktion, abstrakte Klasse, Operatorüberladung, friend-Funktion):

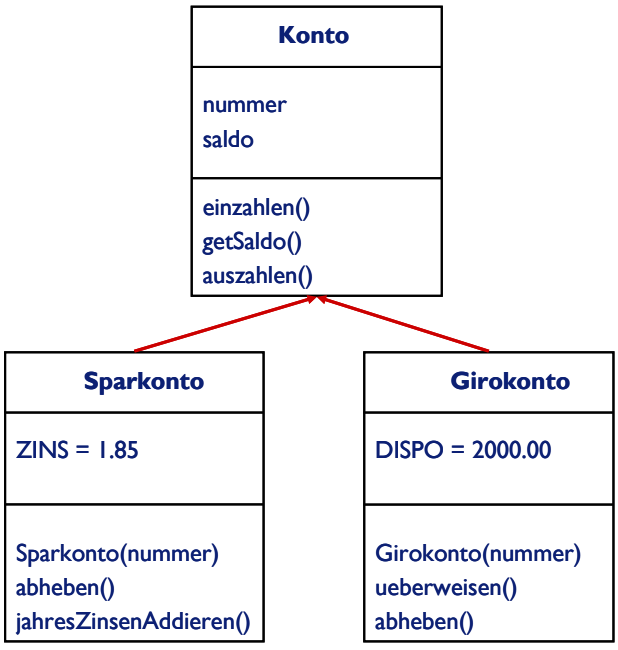
Das unten stehende Diagramm zeigt eine stark vereinfachte Klassenhierarchie, wie es sie für eine Anwendung im Bankensektor geben könnte. Es gibt dort verschiedene Kontoarten, die einige gleiche Attribute und Methoden haben, sich aber in anderen Dingen unterscheiden. Ihre Aufgabe ist es, dieses Modell zu implementieren und anschließend mit Hilfe einer kleinen Anwendung zu testen. Dabei sind die folgenden Anforderungen zu beachten:

- Das Guthaben (Saldo) des ***Girokontos*** darf im Rahmen des Dispokredits (2000 Euro) negativ werden.
- Das Guthaben des ***Sparkontos*** darf 0 Euro nicht unterschreiten.
- Die Methode zur Zinsberechnung des Sparkontos (**`jahresZinsenAddieren()`**) soll die Zinsen für ein Jahr berechnen (saldo * Zinsen / 100) und dem Saldo hinzufügen.
- Beim Überweisen und Abheben soll die entsprechende Summe vom Saldo abgezogen und eine entsprechende Meldung ausgegeben werden. Falls eine Transaktion aufgrund des Kontostandes nicht durchgeführt werden kann, soll dies ausgegeben werden.
- Bei weiteren verwendeten Transaktionen (z.B. einzahlen) soll ebenfalls eine entsprechende Meldung ausgegeben werden.
- In **`Konto`** soll **`abheben()`** als ***abstrakte Funktion*** festgeschrieben sein:

```
virtual void abheben(float summe) = 0;
```

  Abstrakte Funktionen werden verwendet, um in einer Basisklasse ein gemeinsames Interface festzulegen, ohne jedoch bereits etwas über die Implementierung zu sagen. Diese muss dann in den abgeleiteten Klassen nachgeholt werden.

  Hinweis: Eine Klasse mit mindestens einer abstrakten Funktion nennt man abstrakte Klasse. Von einer solchen abstrakten Klasse kann kein Objekt instanziiert werden (weil die Definition der abstrakten Funktion(en) fehlt). Wird in einer abgeleiteten Klasse die abstrakte Funktion definiert (man sagt *überschrieben*), kann von dieser ein Objekt instanziiert werden.



a) Implementieren Sie die angegebene Klassenhierarchie.

b) Implementieren Sie für die beiden abgeleiteten Klassen jeweils eine geeignete Überladung des Ausgabeoperators „<<“, die Sie als **`friend`** deklarieren.

c) Testen Sie Ihr Programm mit folgender Anwendung. Welche Zeilen müssen auskommentiert werden, damit das Programm kompiliert (mit Begründung!)?

Die Lösung soll die Implementierung und die Bildschirmausgabe des Programms umfassen.

```
int main(int argc, char *argv[])
{
    // ein paar Experimente mit einem Girokonto
    // ----------------------------------------
    // Konto anlegen
    Girokonto girokonto1(111111);

    // Geld einzahlen
    girokonto1.einzahlen(1344.99);
    cout << girokonto1 << endl;

    // erfolgreich Geld abheben
    girokonto1.abheben(2000);
    cout << girokonto1 << endl << endl;

    Girokonto* girokonto2 = new Girokonto(222222);
    cout << *girokonto2 << endl << endl;

    // erfolgreicher Überweisungsversuch
    girokonto1.ueberweisen(850, *girokonto2);
    cout << girokonto1 << endl;
    cout << *girokonto2 << endl << endl;

    // vergeblicher Überweisungsversuch
    girokonto1.ueberweisen(5000, *girokonto2);
    cout << girokonto1 << endl;
    cout << *girokonto2 << endl << endl;

    cout << endl;

    // ein paar Experimente mit einem Konto
    // ------------------------------------
    // Konto anlegen
    Konto k(12345);

    // Geld einzahlen
    k.einzahlen( 1000.0 );
    cout << k << endl;

    // Geld auszahlen
    k.auszahlen( 500.0 );
    cout << k << endl << endl;
```

```
    // ein paar Experimente mit einem Sparkonto
    // ----------------------------------------
    // Konto anlegen
    Sparkonto sparkonto(333333);

    // Geld einzahlen
    sparkonto.einzahlen(2346.88);
    cout << sparkonto << endl << endl;

    // erfolgreich Geld abheben
    sparkonto.abheben(2000);
    cout << sparkonto << endl << endl;

    // Zins-Tag!
    sparkonto.jahresZinsenAddieren();
    cout << sparkonto << endl;

    return EXIT_SUCCESS;
}
```