FACH:	**C++ für Fortgeschrittene**	**NAME:**	
DATUM:	Mittwoch, 14. März 2007		
ZEIT:	14:00 – 17:00	**SEMESTER:**	
PRÜFER:	Erben		
		STUDIENGANG:	☐ Dipl. Math. (benotet) ☐ anderer (unbenotet)

ANLAGEN:
- Definition und Implementation einiger Klassen (**bereits vorab ausgegeben**).
- Alle Aufgaben beziehen sich auf diese Klassen inklusive der als Bibliothek vorgegebenen Klasse `Polynom`.

HILFSMITTEL:
- Zuvor termingerecht abgegebene Übungen und Studienarbeiten ohne (wesentliche) handschriftliche Ergänzungen
- Handschriftliche Ergänzungen auf den Vorderseiten der vorab ausgegebenen Anlagen

UNBEDINGT BEACHTEN:
- Tragen Sie **jetzt gleich** auf dieser Seite Ihren **Name** und ihr **Semester** ein und kreuzen Sie Ihren **Studiengang** an. Bearbeitungen ohne diese vorherige Kennzeichnung sind ungültig.
- Führen Sie die Bearbeitung direkt auf diesen Aufgabenblättern in den dafür vorgesehenen Rahmen aus. Der dort vorhandene Platz sollte ausreichen.
- Bei dem in den Aufgaben angegebenen Quellcode handelt es sich meist um Auszüge, welche sinnvoll ergänzt zu denken sind. Insbesondere trifft dies auf das Einbinden von benötigten Headern zu. Auch bei Ihren Bearbeitungen brauchen Sie nirgends die benötigten Header anzugeben.
- Allgemeine Programmier- und Design-Regeln müssen eingehalten werden. Kommentare sind aber generell nicht verlangt.

EMPFEHLUNGEN ZUR BEARBEITUNG:
- Lesen Sie die Aufgaben und Aufgabenteile bitte in der vorgegebenen Reihenfolge durch. An manchen Stellen ist dies für das Verständnis der Aufgabenstellung wichtig.
- Einzelne Bearbeitungen können aber zurückgestellt werden. Die vollständige Lösung einer Aufgabe ist für die folgenden Aufgaben nicht erforderlich.

Aufgabe 1: (24 Punkte)

Diese Aufgabe verwendet außer der Bibliotheksklasse `Polynom` keine Klassen der Anlage.

a) Die Tschebyscheff-Polynome $T_k(x)$ mit $k \in \mathbb{N}_0$ genügen der Rekursionsvorschrift

$$T_0(x) = 1\ , \qquad T_1(x) = x\ , \qquad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \qquad (k = 2, 3, 4 \ldots)$$

Schreiben Sie eine globale Funktion, welche das Tschebyscheff-Polynom eines gegebenen Grades `grad` liefert. Sichern Sie zu, dass der Aufruf nicht mit negativem Grad erfolgt.

```
Polynom tschebyscheffPolynom(int grad) {
        assert(grad >= 0);
        if (grad == 0) return 1.;
        if (grad == 1) return x;
        Polynom p1 = tschebyscheffPolynom(grad – 2);
        Polynom p2 = tschebyscheffPolynom(grad – 1);
        Polynom erg = 2. * x * p2 – p1;
        return erg;
}
```

b) Schreiben Sie ein Hauptprogramm, welches (mehrere) ganze Zahlen aus einer Datei namens `eingabe.txt` einliest und zu jeder dieser Zahlen das Tschebyscheff-Polynom dieses Grades in einer eigenen Zeile auf den Bildschirm ausgibt. Es brauchen keinerlei Fehler (z.B. Datei nicht vorhanden) abgefangen zu werden.

```
void main() {
      ifstream datei("eingabe.txt");
      int grad;
      while (datei >> grad)
            cout << tschebyscheffPolynom(grad) << endl;
}
```

Aufgabe 2: (20 Punkte)

Diese Aufgabe beschäftigt sich mit dem expliziten Konstruktor der Klasse `PolynomFunktion` aus der Anlage.

a) Der Konstruktor liefert zwar das gewünschte Resultat, die Implementierung entspricht aber nicht den Empfehlungen. Verbessern Sie diese Implementierung:

```
PolynomFunktion::PolynomFunktion(const Polynom& p)
                : mPolynom(p) {
}
```

b) Der Unterschied liegt in den aufgerufenen Methoden der Klasse `Polynom`. Entscheiden Sie, welche der folgenden Methoden in der ursprünglichen Implementierung und in Ihrer Verbesserung gerufen werden. Geben Sie zusätzlich an, welche der aufgeführten Methoden regular sind.

Methode der Klasse `Polynom`	gerufen in Klasse der Anlage		gerufen in obiger Verbesserung		Die Methode ist regular	
	ja	nein	ja	nein	ja	nein
Default-Konstruktor	✓			✓		✓
Copy-Konstruktor		✓	✓		✓	
Zuweisungs-Operator	✓			✓	✓	
Destruktor		✓		✓	✓	
Gleichheits-Operator		✓		✓	✓	

Aufgabe 3: (20 Punkte)

Die Aufgabe bezieht sich auf den virtuellen Mechanismus der Klassenhierarchie.

a) Betrachten Sie das Code-Fragments

```
PolynomFunktion p = -5.1;
Betrag b(p);
```

```
cout << "p = " << p << ", b = " << b << endl;
```

Was wird in der markierten Zeile ausgegeben? Wie sähe die Ausgabe aus, wenn die Methode `ausgebenAuf` in der Definition der Klasse `Funktion` nicht `virtual` deklariert würde? (Alle anderen Klassen bleiben unverändert.)

mit `virtual`	**`p = -5.1, b = |-5.1|`**
ohne `virtual`	**`p = ???, b = ???`**

b) Denken Sie sich obigen Code wie folgt fortgesetzt

```
Funktion& f = p;
cout << p(0) << " / " << b(0) << " / " << f(0) << endl;
```

Was wird in der markierten Zeile ausgegeben? Wie sähe die Ausgabe aus, wenn der Operator `operator()` in der Definition der Klasse `Funktion` nicht `virtual` deklariert würde? (Alle anderen Klassen bleiben unverändert.)

mit `virtual`	**`-5.1 / 5.1 / -5.1`**
ohne `virtual`	**`-5.1 / 0 / 0`**

Aufgabe 4: (20 Punkte)

In dieser Aufgabe geht es um einen Fehler in der Basisklasse `Funktion`.

```
Betrag* f = new Betrag();
delete f;
```

funktioniert einwandfrei. Nach der minimalen Änderung

```
Funktion* f = new Betrag();
delete f;
```

wird jedoch der in der Klasse `Dekorierer` dynamisch allokierte Speicher nicht wieder freigegeben.

a) Um den Fehler zu beheben, muss eine implizite Methode der Klasse `Funktion` explizit definiert werden. Geben Sie an, welche Zeile hierzu in den `public`-Teil der Klassendefinition eingefügt werden muss.

```
virtual ~Funktion();
```

b) Geben Sie die Implementierung der Methode an.

```
Funktion::~Funktion()
{
```

```
}
```

c) Prüfen Sie, ob das System ohne die Fehlerbehebung auch im Falle von

```
Dekorierer* f = new Betrag();
delete f;
```

blutet. Begründen Sie präzise das Ergebnis.

```
Das System blutet nicht.
delete ruft den Destruktur des statischen Typs von f,
also von Dekorierer.
Dieser Destruktur gibt den Speicher frei.
```

Aufgabe 5: (24 Punkte)

Die Klassen sollen um eine Methode namens `istPolynom` erweitert werden. Die Methode soll `true` liefern, wenn die `Funktion` (mathematisch!) als Polynom darstellbar ist und `false`, falls dies falsch oder zumindest unklar ist.

a) Die Methode soll in der Basisklasse `Funktion` abstrakt sein. Wie muss die Methode in der Klassendefinition von `Funktion` deklariert werden?

```
virtual bool istPolynom() const = 0;
```

b) In der Klasse `PolynomFunktion` wird die Methode dann in naheliegender Weise überschrieben. Geben Sie die zugehörige Implementierung an.

```
bool PolynomFunktion::istPolynom() const {
    return true;
}
```

c) Eine Spiegelung ist ein Polynom, falls die zu spiegelnde Funktion eines ist. Die Klasse `SpiegelungHorizontal` kann dieses Verhalten nicht selbst erreichen, sondern ist auf die Erweiterung einer anderen Klasse angewiesen. Warum?

Die die zu spiegelnde Funktion (Variable mFunktion)
ist in der Klasse Dekorierer private.

d) Geben Sie die entsprechende Implementierung (in dieser anderen Klasse) an.

```
bool Dekorierer::istPolynom() const {
    return mFunktion->istPolynom();
}
```

Aufgabe 6: (24 Punkte)

In dieser Aufgabe geht es um einfache Felder und Zeiger.

a) Prüfen Sie, ob die Anweisungen in der ersten Spalte der folgenden Tabelle fehlerfrei compilierbar sind. Geben Sie in diesem Falle in der letzten Spalte die Ausgabe von

```
cout << f[0] << " / " << f[1] << endl;
```

an. Im Falle eines Compile-Fehlers geben Sie bitte in der letzten Spalte eine (ganz) kurze Begründung an.

fehlerfrei compilierbar?	ja	nein	bei ja: Ausgabe bei nein: kurze Begründung
`Funktion f[87];`		✓	**Klasse ist abstrakt**
`Dekorierer f[87];`		✓	**kein Default-Konstruktor**
`PolynomFunktion f[87];`		✓	**kein Default-Konstruktor**
`Betrag f[87];`	✓		**|x| / |x|**
`Kehrwert f[87];`		✓	**kein Default-Konstruktor**

b) Der Code

```
Funktion *g = new PolynomFunktion(1.);
```

```
for (int i = 0; i < 2; i++) {
     cout << *g << endl;
     g = new PolynomFunktion(2.);
}
```

soll um die Freigabe des gesamten allokierten Speichers ergänzt werden. Außerdem sollen die beiden markierten Stellen so verändert werden, dass die Ausgabe

```
|x|
1/(x)
```

erfolgt. Geben Sie den gesamten neuen Code an:

```
Funktion *g = new Betrag();
for (int i = 0; i < 2; i++) {
     cout << *g << endl;
     delete g;
     g = new Kehrwert(PolynomFunktion(x));
}
delete g;
```

Aufgabe 7: (24 Punkte)

Es geht darum, bei überladenen Methoden die tatsächlich aufgerufene zu erkennen. Der Übersichtlichkeit halber wird statt des Typs `SpiegelungHorizontal` die Kurzform `Spiegelung` verwendet. Dies kann auch formal für den Compiler sichergestellt werden durch

```
typedef SpiegelungHorizontal Spiegelung;
```

a) Es soll die Ausgabe mehrerer Code-Fragmente untersucht werden, welche alle nach dem Schema

```
PolynomFunktion p = 5 * x + 1;
Kehrwert f = p;
Funktion *g = new Spiegelung(f);
cout << *g << endl;
```

aufgebaut sind. Die Fragmente unterscheiden sich nur an den beiden markierten Stellen. Was dort jeweils genau zu stehen hat, wird durch die Spalten und Zeilen der nachstehende Tabelle festgelegt. Tragen Sie in die Tabelle ein, welche Ausgabe jeweils in der letzten Zeile des obigen Codes erfolgt.

	`Kehrwert` `f = p;`	`Spiegelung` `f = p;`
`Funktion *g = &f;`	`1/(5x+1)`	`0-(5x+1)`
`Funktion *g = new Kehrwert(f);`	`1/(5x+1)`	`1/(0-(5x+1))`
`Funktion *g = new Spiegelung(f);`	`0-(1/(5x+1))`	`0-(5x+1)`
`Funktion *g = new Spiegelung(f,1.);`	`1-(1/(5x+1))`	`1-(0-(5x+1))`

b) Das zugrunde liegende Schema ist nun

```
PolynomFunktion p = x * (x + 1);
Funktion *f = new Kehrwert(p);
Funktion *g = new Spiegelung(*f);
cout << *g << endl;
```

Tragen Sie wieder in die Tabelle ein, welche Ausgabe jeweils in der letzten Zeile dieses Codes erfolgt.

	`Funktion *f = new Kehrwert(p);`	`Funktion *f = new Spiegelung(p);`
`Funktion *g = new Kehrwert(*f);`	`1/(1/(x^2+x))`	`1/(0-(x^2+x))`
`Funktion *g = new Spiegelung(*f);`	`0-(1/(x^2+x))`	`0-(0-(x^2+x))`
`Funktion *g = new Dekorierer(*f);`	`1/(x^2+x)`	`0-(x^2+x)`

Aufgabe 8: (24 Punkte)

In dieser Aufgabe geht es um die Untersuchung des Verhaltens von vorgelegtem Code.

a) Betrachten Sie zunächst die Zeilen

```
PolynomFunktion p = 1.;
p = x;
```

Geben Sie alle (expliziten und impliziten) Methoden der Klassen aus der Anlage (ohne die Klasse `Polynom`) an, die (explizit oder implizit) gerufen werden. Beachten Sie, dass meist – eventuell implizit – Methoden von Basisklassen gerufen werden. Auch diese sind verlangt. Achten Sie darauf, dass präzise geklärt ist, welche Methode in welcher Klasse Sie meinen.

Im ersten Statement wird gerufen:

```
Konstruktor PolynomFunktion(const Polynom&);
Default-Konstruktor von Funktion
```

Im zweiten Statement wird gerufen:

```
Konstruktor PolynomFunktion(const Polynom&);
Default-Konstruktor von Funktion
Zuweisungs-Operator von PolynomFunktion
Zuweisungs-Operator von Funktion
Destruktor von PolynomFunktion
Destruktor von Funktion
```

b) Die harmlos erscheinende Frage nach der Ausgabe von

```
PolynomFunktion p = 2.;
Dekorierer* f = new Betrag[3];
f[0] = Betrag(p);
f[1] = Kehrwert(p);
cout << f[0] << " / " << f[1] << " / " << f[2] << endl;
```

hat es in sich. Eine korrekte Beantwortung ist wohl nur möglich durch eine sorgfältige Untersuchung, welche Zuweisung in den markierten Statements gerufen wird und was diese Zuweisung genau macht. Was wird nun ausgegeben?

```
|2| / |2| / |x|
```