RECHEN- UND
KOMMUNIKATIONSZENTRUM DER RWTH AACHEN

Jürgen Dietel, Willi Hanrath

Klausur zum C++-Kurs für MaTAs im SS 2006
am 15. März 2006
Dauer: 2 Stunden

Name:
Vorname:
Kenn-Nummer:
Matrikelnummer:
Unterschrift:

			max. Punktzahl
Aufgabe	1)		(16)
Aufgabe	2)		(6)
Aufgabe	3)		(6)
Aufgabe	4)		(14)
Aufgabe	5)		(16)
Aufgabe	6)		(9)
Aufgabe	7)		(17)
Aufgabe	8)		(16)
maximale Summe:			(100)

Gesamtpunkte:	Note:

Aufgabe 1: *16 Punkte*

Fragen zu Verschiedenem.

a) Was ist bei der Einbindung alter, von C geerbter Standardheaderdateien (z. B. `stdio.h`) in einer C++-Quelldatei zu beachten? (2 Punkte)

b) Was ist in einer C++-Quelldatei bei der Deklaration von Funktionen zu beachten, die aus einer durch einen C-Compiler erstellten Objektdatei stammen? (2 Punkte)

c) Welche Regeln sind bei der Verwendung von **`const`**-Objekten zu beachten? (6 Punkte)

d) Was ist bei konstanten Memberfunktionen zu beachten? (2 Punkte)

e) Werden Standard-Parameter einer Funktion bei der Funktions**definition** oder der Funktions**deklaration** angegeben? (2 Punkte)

f) Ist der Zugriffsschutz **`public`** bei Klassendefinition durch **`struct`** oder durch **`class`** voreingestellt? (2 Punkte)

Aufgabe 2: *6 Punkte*

Fragen zu Namensbereichen. Beantworten Sie folgende Fragen zum Thema *Namensbereich*:

a) Dürfen eine globale Funktion und eine globale Variable in derselben Quelldatei den gleichen Namen tragen?
ja ☐ nein ☐

b) Kann durch das Schlüsselwort `static` die Bekanntheit eines globalen Namens (Funktion oder Variable) auf einen Quelltext eingeschränkt werden?
ja ☐ nein ☐

c) Kann man bei der Zuordnung einer Variablen zu einem Namensbereich zwischen Deklaration und Definition unterscheiden?
ja ☐ nein ☐

d) Kann nach Definition einer lokalen Variablen `g` in `main()` dort auch noch eine gleichnamige globale Variable angesprochen werden?
ja ☐ nein ☐

e) Kann durch `A.g = 5;` der Variablen `g` aus dem Namensbereich `A` ein neuer Wert zugewiesen werden?
ja ☐ nein ☐

f) Werden durch `using namespace A;` alle Namen aus dem Namensbereich `A` so zur Verfügung gestellt, daß man sie bei ihrer Verwendung nicht qualifizieren muß?
ja ☐ nein ☐

Aufgabe 3:

6 Punkte

Zugriffsabschnitte. Betrachten Sie den C++-Quelltext in Listing 1 und Listing 2, und notieren Sie, welche Zugriffe auf die Membervariablen der Klasse `A` in den Funktionen `f`, `g`, `h`, `r`, `s` und `main` erlaubt sind und welche nicht, indem Sie zu jeder betroffenen Zeilennummer angeben, ob der entsprechende Zugriff erlaubt ist oder nicht. (Abgesehen von einigen der genannten Zugriffe enthält der Quelltext keine Fehler, liefert also nach Beseitigung der nicht erlaubten Zugriffe ein lauffähiges Programm.)

```
1  #include <iostream>
2
3  class A {
4    private:
5      int i;
6    protected:
7      int j;
8    public:
9      int k;
10     A(int i0, int j0, int k0)
11     : i(i0), j(j0), k(k0) { }
12     void f() {
13       std::cout << std::endl;
14       std::cout << "i=" << i << std::endl;
15       std::cout << "j=" << j << std::endl;
16       std::cout << "k=" << k << std::endl;
17     }
18     friend void g(A);
19     friend class D;
20 };
21
22 void g(A a) {
23   a.i = 21;
24   a.j = 22;
25   a.k = 23;
26   a.f();
27 }
28
29 class B {
30   public:
31     static void h(A a) {
32       a.i = 31;
33       a.j = 32;
34       a.k = 33;
35       a.f();
36     }
37 };
```

Listing 1: fehlerhafter C++-Quelltext (1/2)

```
39 class C: A {
40   public:
41     C() : A(1,2,3) { }
42     void r() {
43       i = 41;
44       j = 42;
45       k = 43;
46       f();
47     }
48 };
49
50 class D {
51   public:
52     static void s(A a) {
53       a.i = 51;
54       a.j = 52;
55       a.k = 53;
56       a.f();
57     }
58 };
59
60 int main() {
61   A a(1,2,3);
62   a.f();
63
64   a.i = 11;
65   a.j = 12;
66   a.k = 13;
67   a.f();
68
69   g(a);
70   B::h(a);
71   C c;
72   c.r();
73   D::s(a);
74   return 0;
75 }
```

Listing 2: fehlerhafter C++-Quelltext (2/2)

a) in `f`:

Zeile 14:	erlaubt ☐	nicht erlaubt ☐
Zeile 15:	erlaubt ☐	nicht erlaubt ☐
Zeile 16:	erlaubt ☐	nicht erlaubt ☐

b) in `g`:

Zeile 23:	erlaubt ☐	nicht erlaubt ☐
Zeile 24:	erlaubt ☐	nicht erlaubt ☐
Zeile 25:	erlaubt ☐	nicht erlaubt ☐

c) in `h`:

Zeile 32:	erlaubt ☐	nicht erlaubt ☐
Zeile 33:	erlaubt ☐	nicht erlaubt ☐
Zeile 34:	erlaubt ☐	nicht erlaubt ☐

d) in `r`:

Zeile 43:	erlaubt ☐	nicht erlaubt ☐
Zeile 44:	erlaubt ☐	nicht erlaubt ☐
Zeile 45:	erlaubt ☐	nicht erlaubt ☐

e) in `s`:

Zeile 53:	erlaubt ☐	nicht erlaubt ☐
Zeile 54:	erlaubt ☐	nicht erlaubt ☐
Zeile 55:	erlaubt ☐	nicht erlaubt ☐

f) in `main`:

Zeile 64:	erlaubt ☐	nicht erlaubt ☐
Zeile 65:	erlaubt ☐	nicht erlaubt ☐
Zeile 66:	erlaubt ☐	nicht erlaubt ☐

Aufgabe 4: *14 Punkte*

Fragen zu Zeigern.

a) Es seien die wie folgt deklarierten Funktionen gegeben: (8 Punkte)

```
int g(int);
int* h(int);
```

Welche der folgenden Aufrufe dieser Funktionen sind korrekt bzw. nicht korrekt?

i)	`i = ++g(7);`	korrekt: ☐	nicht korrekt: ☐
ii)	`i = ++(*h(7));`	korrekt: ☐	nicht korrekt: ☐
iii)	`int* p = &g(7);`	korrekt: ☐	nicht korrekt: ☐
iv)	`int* p = h(7);`	korrekt: ☐	nicht korrekt: ☐
v)	`g(7) = 7;`	korrekt: ☐	nicht korrekt: ☐
vi)	`*h(7) = 7;`	korrekt: ☐	nicht korrekt: ☐
vii)	`int i = g(*h(7));`	korrekt: ☐	nicht korrekt: ☐
viii)	`int i = *h(g(7));`	korrekt: ☐	nicht korrekt: ☐

b) Es seien die wie folgt deklarierten Funktionen gegeben: (6 Punkte)

```
void g1(int);
void g2(int*);
void g3(const int*);
```

und diese Variablendefinition:

```
const int i = 1;
```

Welche der folgenden Aufrufe dieser Funktionen sind korrekt bzw. nicht korrekt?

i)	`g1(0);`	korrekt: ☐	nicht korrekt: ☐
ii)	`g1(&i);`	korrekt: ☐	nicht korrekt: ☐
iii)	`g2(0);`	korrekt: ☐	nicht korrekt: ☐
iv)	`g2(&i);`	korrekt: ☐	nicht korrekt: ☐
v)	`g3(0);`	korrekt: ☐	nicht korrekt: ☐
vi)	`g3(&i);`	korrekt: ☐	nicht korrekt: ☐

Aufgabe 5: *16 Punkte*

Templates.

a) Verallgemeinern Sie die in Listing 3 gegebene Klasse so, daß in Zukunft statt des fest eingebauten Typs **int** ein frei wählbarer ein- bzw. ausgekellert werden kann, für den die Zuweisung = und der Kopierkonstruktor definiert sind. (8 Punkte)

```
4  class Stack {
5    public:
6      Stack();
7      void push(int wert);
8      int pop(void);
9      class Overflow { };
10     class Underflow { };
11   protected:
12     int keller[100];
13     int sp;
14 };
15
16 Stack::Stack() {
17   sp = 0;
18 }
19
20 void Stack::push(int wert) {
21   if (sp >= 100)
22     throw Overflow();
23   else
24     keller[sp++] = wert;
25 }
26
27 int Stack::pop(void) {
28   if (sp == 0)
29     throw Underflow();
30   else
31     return keller[ --sp];
32 }
```

Listing 3: Klasse `Stack` für den Datentyp **int**

b) Schreiben Sie eine Funktion, die von drei gegebenen Objekten das größte ermittelt und als Ergebnis zurückliefert. (8 Punkte)

Dabei sollen diese drei Objekte der Funktion als Eingabeparameter übergeben werden und vom selben Typ sein, für den alle Vergleichsoperatoren definiert sind.

Aufgabe 6: *9 Punkte*

Konstruktoren.

a) Welchen Rückgabetyp hat ein Konstruktor?

b) Welcher Konstruktor wird implizit vom System zur Verfügung gestellt, wenn eine Klasse keinen explizit definierten Konstruktor enthält?

c) Wie lautet die Signatur des Kopierkonstruktors der Klasse `A`?

d) In welchem Fall ist man gezwungen, den Kopierkonstruktor zu definieren?

e) Wie kann man die Benutzung des Kopierkonstruktors so weit wie möglich verbieten?

f) i) Welchen Ersatz gibt es in Konstruktoren für die Zuweisung von Werten an Membervariablen?

ii) In welchen Fällen ist dieser „Ersatz“ die einzig mögliche Alternative zu einer Zuweisung? Nennen Sie zwei Möglichkeiten.

g) Was geschieht hier (`A` sei eine Klasse)?

```
A* p = new A[99];
```

Aufgabe 7: *17 Punkte*

Komplexe Zahlen. Entwerfen und implementieren Sie (Teile) eine(r) Klasse `Complex` zur Darstellung von komplexen Zahlen (Re, Im) mit einem reellen Realteil Re und Imaginärteil Im. (Real- und Imaginärteil sollen nicht öffentlich zugreifbar sein!)

Genaue Anforderungen, welche Teile der Klasse wie zu implementieren sind:

a) eine komplexe Zahl soll wie folgt erzeugt werden können: (3 Punkte)

- parameterlos: `Complex z;`
  Die komplexe Zahl `z` soll der Zahl 0 entsprechen.
- mit einem **`double`**-Argument: `Complex z(x);`
  Die komplexe Zahl `z` soll dem Paar `(x,0)` entsprechen.
- mit zwei **`double`**-Argumenten: `Complex z(x,y);`
  Die komplexe Zahl `z` soll dem Paar `(x,y)` entsprechen.

b) Operatorfunktion `-` zur Generierung des negativen Wertes einer komplexen Zahl, (3 Punkte)

c) globale Operatorfunktion `+` zur Addition zweier komplexer Zahlen, (3 Punkte)

d) Operatorfunktion `*` zur Multiplikation zweier komplexer Zahlen, realisiert als Memberfunktion, (3 Punkte)

e) Konversionsoperator zur Umwandlung einer komplexen Zahl in einen **`double`**-Wert durch Berechnung ihres Betrags, (2 Punkte)

f) Ausgabeoperator `<<` zur Ausgabe einer komplexen Zahl auf den Bildschirm. (3 Punkte)

Aufgabe 8: *16 Punkte*

Integrierbare Funktionen.

a) Definieren Sie eine abstrakte Basisklasse `intbare_Funktion` zur Handhabung integrierbarer mathematischer Funktionen. Diese soll (neben den für abstrakte Basisklassen notwendigen Funktionen) über die öffentlichen, rein virtuellen Memberfunktionen (4 Punkte)

```
double auswerten(double);
```

bzw.

```
intbare_Funktion* stammfunktion();
```

zur Auswertung der (mathematischen) Funktion an der übergebenen Stelle bzw. zur Erzeugung *einer* (mathematischen) Stammfunktion verfügen.

b) Leiten Sie aus dieser Basisklasse `intbare_Funktion` die Klasse `polynom` zur Handhabung von Polynomen ab. (10 Punkte)

Die Klasse `polynom` soll `int grad;` und `double *koeff;` als (**protected**) Komponenten besitzen, wobei `grad` der Grad des Polynoms ist und `koeff` auf ein dynamisch anzulegendes **double**-Feld der Länge `grad+1` zeigt, in dem die Koeffizienten des Polynoms abgespeichert sind. Definieren Sie die für Klassen mit dynamische Komponenten erforderlichen Funktionen, einen naheliegenden Konstruktor `polynom(int, double*)` und passende Realisierungen der Funktionen `auswerten` und `stammfunktion`.

c) Leiten Sie aus der Klasse `intbare_Funktion` eine Klasse `gebr_rat_Fkt` für gebrochen rationale Funktionen her. (Gebrochen rationale Funktionen sind „Brüche“, wobei Zähler und Nenner jeweils Polynome sind!) (6 Punkte)

Objekte dieser Klasse haben somit zwei (**protected**) Komponenten vom Typ `polynom`. Als Memberfunktion sollen ein Konstruktor

```
gebr_rat_Fkt(polynom& zaehler, polynom& nenner);
```

und Realisierungen von `auswerten` und `stammfunktion` (und was sonst noch unbedingt erforderlich ist!) vorhanden sein.

Hinweis:

Falls in abgeleiteten konkreten Klassen keine allgemeingültige Formel für eine (mathematische) Stammfunktion existiert, soll `stammfunktion` den Nullzeiger liefern.