Prof. Dr. Stefan Brass, Dr. Henning Thielemann
Institut für Informatik
Martin-Luther-Universität Halle-Wittenberg

15. September 2009

Vorlesung „Objektorientierte Programmierung“
— Klausur —

Name: _____

Studiengang: _____

Matrikelnummer: _____

(Diese Daten werden zur Ausstellung des Leistungsnachweises benötigt.)

Aufgabe	Punkte	Max. Punkte	Zeit
1 (Programmierung: Klasse)		8	15 min
2 (Programmierung: Feld)		6	10 min
3 (Logischer Ausdruck, Toter Code)		3	10 min
4 (Globale/lokale Var., Call by Value/Ref.)		6	10 min
5 (Zeiger, C-String)		1	5 min
6 (Rekursion)		2	5 min
7 (Vererbung)		1	5 min
8 (Fehlertypen)		3	5 min
9 (Fehlersuche)		3	10 min
Summe		33	75 min

☐ Ich fühle mich gesundheitlich in der Lage, diese Prüfung abzulegen. (Bitte sprechen Sie mit dem Aufsichtspersonal, falls Sie sich krank fühlen.)

☐ Für den Fall, daß ich nicht korrekt zu dieser Prüfung angemeldet sein sollte, erkläre ich mich sowohl damit einverstanden, daß ich rückwirkend angemeldet werde, als auch damit, daß diese Prüfung nicht zählt (Entscheidung des Dozenten).

Unterschrift: _____

Hinweise:

- Bearbeitungsdauer: 90 Minuten
- Skript, Bücher, Notizen sind erlaubt.
  Notebooks, PDAs usw. dürfen nicht verwendet werden. Mobiltelefone bitte ausschalten (oder mit Aufsicht besprechen).
- Die Klausur hat 14 Seiten. Bitte prüfen Sie die Vollständigkeit.
- Verwenden Sie weder Rot- noch Bleistift zum Bearbeiten der Aufgaben.
- Bitte benutzen Sie den vorgegebenen Platz. Wenn Sie auf die Rückseite ausweichen müssen, markieren Sie klar, dass es eine Fortsetzung gibt.
- Tauschen Sie keinesfalls irgendwelche Dinge mit den Nachbarn aus. Rufen Sie notfalls eine Aufsichtsperson zur Kontrolle.
- Fragen Sie, wenn Ihnen eine Aufgabe nicht klar ist!
- Zum (garantierten) Bestehen benötigen Sie 60% der Punkte (16.5/27). Die Grenze wird möglicherweise gesenkt.

Zum Nachschlagen:

- Tabelle mit den Prioritätsstufen der Operatoren:

18	`::`	Gültigkeitsbereich
17	`++` (Postfix), `.`, `->`, `[]`, `f()`, ...	Postfix-Operatoren
16	`-` (unär), `!`, `*` (deref), `++` (Präfix), ...	Präfix-Operatoren
15	`.*`, `->*`	Zeiger auf Komp.
14	`*`, `/`, `%`	Multiplikation etc.
13	`+`, `-`	Addition, Subtraktion
12	`<<`, `>>`	Shift etc.
11	`<`, `<=`, `>`, `>=`	kleiner etc.
10	`==`,`!=`	gleich, verschieden
9	`&`	Bit-und
8	`^`	Bit-xor
7	`|`	Bit-oder
6	`&&`	und
5	`||`	oder
4	`?:`	Bedingter Ausdruck
3	`=`, `+=`, `-=`, `*=`, `/=`, ...	Zuweisungen
2	`throw`	Exception auslösen
1	`,`	Sequenz

  Bei gleicher Priorität sind alle Operatoren (außer Präfixoperatoren und Zuweisungen) implizit von links geklammert (linksassoziativ).

- Bei der Ganzzahl-Division ist für positive Eingabewerte garantiert, daß abgerundet wird, z.B. liefert `8/3` das Ergebnis `2`.

Aufgabe 1 (Programmierung: Klasse) 8 Punkte

Ein Feuerwerker besitzt Mörserkästen, aus denen er bei Großfeuerwerken Feuerwerksbomben abschießt. Ein Mörserkasten enthält n Rohre (z.B. 5) von Kaliber k (Durchmesser der zugehörigen Feuerwerksbomben, z.B. 100mm, das Rohr hat einen ein wenig größeren Innendurchmesser).
Zur Erläuterung: In jedes Rohr kommt eine Feuerwerksbombe mit einer Treibladung. Sie wird bei dem Feuerwerk aus dem Rohr senkrecht nach oben geschossen, genau wie bei einer Kanone. Mehr oder weniger am höchsten Punkt ihrer Flugbahn zerplatzt sie dann in Leuchtsterne oder andere Effekte. Bei Großfeuerwerken werden Hunderte solcher Bomben abgeschossen, entsprechend viele Rohre müssen vorher aufgebaut werden. Dies geschieht durch Verbinden vieler Mörserkästen zu einem oder mehreren Blöcken.
Es ist für den Feuerwerker außerdem wichtig, über die Anzahl der Verwendungen eines Mörserkastens Buch zu führen (z.B. für gelegentliche, besonders gründliche Prüfungen und für Abschreibungszwecke). Dazu soll jedes Objekt einen Zähler für diese Anzahl enthalten.

Definieren Sie eine Klasse `moerserkasten`, wobei der Konstruktor die beiden Parameter n und k haben soll. Ein Objekt von diesem Typ wird z.B. erzeugt mittels

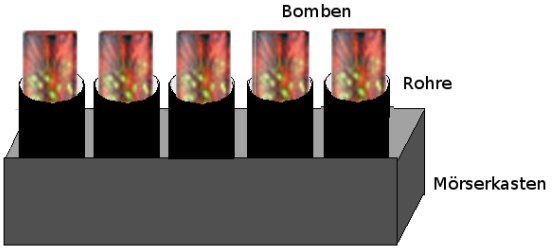
```
moerserkasten x(5, 100); // 5 Rohre mit Kaliber 100mm
```

Der Konstruktor muss natürlich auch den Zähler für die Anzahl der Benutzungen des Kastens auf 0 initialisieren. Ausser dem Konstruktor soll die Klasse folgende Methoden haben:

```
int anzahl_rohre(); // Liefert die Anzahl der Rohre des Kastens,
                    // im Beispiel waere x.anzahl_rohre() == 5
int kaliber();      // Liefert Kaliber der Rohre des Kastens,
                    // im Beispiel 100
int verwendungen(); // Wie haeufig wurde der Kasten bereits verwendet?
void benutze();     // Inkrementiert den Zaehler fuer die Anzahl
                    // der Verwendungen.
```

Der Konstruktor und diese vier Methoden soll die vollständige Schnittstelle der Klasse sein, es darf keine weiteren public Methoden oder Attribute geben.

Es ist Platz für die Lösung auf der nächsten Seite. Beachten Sie, dass auch für ein fehlendes oder falsches Semikolon ein halber Punkt abgezogen werden kann. Versuchen Sie also, Syntaxfehler zu vermeiden. Bemühen Sie sich außerdem um Verständlichkeit und guten Programmierstil. Es können auch für schlechten Stil Punkte abgezogen werden!



Platz zur Lösung von Aufgabe 1

Aufgabe 2 (Programmierung: Feld) 6 Punkte

Gegeben seien die zwei Klassen `datum` und `termine`.

```
class datum {
   private:
      int tag, monat, jahr;

   public:
      void setze_datum (int t, int m, int j) {
         tag = t; monat = m; jahr = j;
      }

      void hole_datum (int &t, int &m, int &j) {
         t = tag; m = monat; j = jahr;
      }
};

class termine {
   private:
      datum *alle_termine;  // Feld, welches alle Termine enthaelt
      int letzter_termin;   // Position des naechsten freien Platzes im Feld
      int max_termine;      // maximale Anzahl der Termine die gespeichert
                            // werden koennen
   public:
      termine (int anzahl) {
         letzter_termin = 0;
         max_termine = anzahl;
         alle_termine = new datum [max_termine];
      }

      bool neuer_termin (int t, int m, int j);
};
```

Die Methode `neuer_termin` soll ein neues Datum in den Terminkalender des Feuerwerkers eintragen. Sie soll dabei folgendes beachten:

- Falls der Terminkalender voll ist, soll die Methode `false` zurückliefern.
- Falls der Termin bereits existiert, soll er nicht eingetragen werden. Die Methode soll jedoch `true` zurückliefern.
- Falls noch Platz im Terminkalender ist und der Termin noch nicht existiert, so soll er in den Terminkalender eingetragen werden und die Methode soll `true` zurückgeben.

Implementieren Sie die Methode `neuer_termin`.

Es ist Platz für die Lösung auf der nächsten Seite. Beachten Sie, daß auch für ein fehlendes oder falsches Semikolon ein halber Punkt abgezogen werden kann. Versuchen Sie also, Syntaxfehler zu vermeiden. Bemühen Sie sich außerdem um Verständlichkeit und guten Programmierstil. Es können auch für schlechten Stil Punkte abgezogen werden!

Platz zur Lösung von Aufgabe 2

Aufgabe 3 (Logischer Ausdruck, Toter Code) 3 Punkte

Im Folgenden ist eine Implementierung einer Funktion angegeben, welche ermittelt, ob ein Jahr im derzeit angewendeten Gregorianischen Kalender ein Schaltjahr ist. Die Regeln für Schaltjahre in diesem Kalender sind:

- Jedes durch 4 teilbare Jahr ist ein Schaltjahr.
- Ausnahme: Alle durch 100 teilbaren Jahre (Säkularjahre) sind keine Schaltjahre.
- Ausnahme von der Ausnahme: Alle durch 400 teilbaren Jahre sind Schaltjahre.

Beispiele: 2000, 2008 sind Schaltjahre, wohingegen 2009, 1900, 2100 keine Schaltjahre sind.

```
bool is_leap_year (int year) {
    bool leap;

    if (year%1 < 0 || year%1 > 0) {
        leap = false;
    } else {
        leap = false;
    }

    if (year%4 == 0) {
        leap = true;
    } else {
        return false;
    }

    if (year%100 != 0) {
        leap = true;
    } else {
        return false;
    }

    if ((year/400)*400 == year) {
        return true;
    } else {
        return false;
    }
}
```

a) Streichen Sie in obigem Programm die Zeilen, die für den berechneten Wert irrelevant sind, also ohne Änderung der Semantik der Prozedur weggelassen werden können. Das gilt natürlich für Programmcode, der niemals ausgeführt wird, aber auch für berechnete Werte, die später nie verwendet werden.

b) Wie könnte man nun die gesamte Berechnung der obigen Funktion in einem arithmetischen Ausdruck zusammenfassen, also den Rumpf der Prozedur auf eine einzige `return`-Anweisung verkürzen?

Lösung:

```
bool is_leap_year (int year) {
    return  ..................................................... ;
}
```

Aufgabe 4 (Globale/lokale Var., Call by Value/Ref.) 6 Punkte

Gegeben ist folgendes Programm:

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

int n = 2;

// Funktion g
void g (______a,_______b) {
   a = a + b;
   b = a * b;
}

// Funktion f
int f (______i) {
    int n = 4;
    g(n, i);
    i += n;
    return i;
}

int main() {
    int n = 10;
    cout << n + f(n);

    return 0;
}
```

Je nachdem, ob `a`, `b` und `i` als Call by value - oder als Call by referenz - Parameter übergeben werden, ändert sich die Ausgabe des Programmes. In der folgenden Tabelle sind 3 mögliche Ausgaben aufgeführt, welche das Programm erzeugen kann. Geben Sie zu diesen jeweils den zugehörigen Funktionskopf der Funktionen g und f an, sodass das Programm die vorgegebene Ausgabe liefert.

Beispiel:

Ausgabe:	Funktionskopf `f`	Funktionskopf `g`
48	`int f (int &i)`	`void g (int &a, int b)`

Ausgabe:	Funktionskopf `f`	Funktionskopf `g`
28	`int f (int ___i)`	`void g (int ___a, int ___b)`
308	`int f (int ___i)`	`void g (int ___a, int ___b)`
154	`int f (int ___i)`	`void g (int ___a, int ___b)`

Aufgabe 5 (Zeiger, C-String) 1 Punkt

Was gibt dieses Programm aus?

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    const char *q = "32312";
    const char *p = q;

    int summe = 0;
    while(*p) {
        summe += p - q;
        p++;
    }
    cout << summe << endl;

    return 0;
}
```

Ausgabe: .....

Aufgabe 6 (Rekursion) 2 Punkte

Schreiben Sie die rekursive Funktion `f` in eine nicht-rekursive Funktion um, welche nur eine Schleife benutzt!

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

int n = 1;

void f(int i)
{
    if(i > 0) {
        n*=2;
        f(i-1);
    } else {
        cout << i << '+' << n;
    }
}

int main()
{
    f(8);

    return 0;
}
```

Antwort:

```
void f(int i)
{

 .....

 .....

 .....

 .....

 .....

}
```

Was gibt dieses Programm aus?: .....

Aufgabe 7 (Vererbung) 1 Punkt

Was gibt dieses Programm aus?

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

class C {
    public:
        int f() { return 1; }
        int g() { return 2; }
        virtual int h() { return 4; }
};

class D : public C {
    public:
        virtual int f() { return 8; }
        int g() { return 16; }
        int h() { return 32; }
};

int main()
{
        D d;
        C* c = &d;
        cout << c->f() + c->g() + c->h();

        return 0;
}
```

Ausgabe: .....

Aufgabe 8 (Fehlertypen) 3 Punkte

Welche der folgenden Fehler sind so ernst, aber auch so offensichtlich, dass der Übersetzer eine Fehlermeldung ausgibt und die Programmerzeugung abbricht und welche Fehler führen höchstens zu Warnungen oder bleiben vom Übersetzer gänzlich unentdeckt? Es ist pro Teilaufgabe genau eine Antwort richtig. Wenn Sie nichts ankreuzen, ist der Punkt auf jeden Fall verloren.

a) Variable nicht deklariert.

- ☐ Der Übersetzer meldet einen Fehler und erzeugt kein Programm.
- ☐ Der Übersetzer gibt höchstens eine Warnung aus, erzeugt aber auf jeden Fall ein Programm.

b) Variable deklariert aber nicht benutzt.

- ☐ Der Übersetzer meldet einen Fehler und erzeugt kein Programm.
- ☐ Der Übersetzer gibt höchstens eine Warnung aus, erzeugt aber auf jeden Fall ein Programm.

c) Variable überschreiben, welche mit `const` deklariert wurde.

- ☐ Der Übersetzer meldet einen Fehler und erzeugt kein Programm.
- ☐ Der Übersetzer gibt höchstens eine Warnung aus, erzeugt aber auf jeden Fall ein Programm.

Aufgabe 9 (Fehlersuche) 3 Punkte

Das folgende Programm enthält Fehler. Bitte geben Sie drei dieser Fehler an. – Es ist möglicherweise schwierig, alle zu entdecken. Falls Sie mehr als drei Fehler angeben, werden nur die ersten drei gewertet. – Es gibt keine Extrapunkte! Geben Sie bitte jeweils die Zeilennummer mit an, in der sich der Fehler befindet. Es zählt auch nicht als Fehler, daß das Programm nichts Sinnvolles tut. Direkte Folgefehler zählen auch nicht.

```
 (1)  #include <iostream>
 (2)  using namespace std;
 (3)
 (4)  class C {
 (5)  private:
 (6)          int a;
 (7)          int square() { return a * a; }
 (8)  public:
 (9)          C(int n) { a = n; }
(10)          int add(int i) { return square() + i; }
(11)  };
(12)
(13)  int main()
(14)  {
(15)          C* x = new C("110");
(16)          int C = 42;
(17)          x->a = 23;
(18)          int i = C + x->square();
(19)          (i+1) = 4;
(20)          i += add(27);
(21)          cout << i + x->add(3);
(22)          return 0;
(23)  }
```

1. Zeile: _____

   Begründung: _____

   _____

2. Zeile: _____

   Begründung: _____

   _____

3. Zeile: _____

   Begründung: _____

   _____