Prof. Dr. Stefan Brass
Dipl.-Inform. Annett Thüring
Institut für Informatik
MLU Halle-Wittenberg

31. März 2009

Vorlesung "Objektorientierte Programmierung" — Klausur —

Name: ______________________

Matrikelnummer: ______________________

Studiengang: ______________________

Aufgabe	Punkte	von	Zeit
1 (Programmierung: Klasse)		9	15 min
2 (Programmierung: Array)		6	10 min
3 (Arithmetischer Ausdruck, Toter Code)		2	10 min
4 (Globale/Lokale Var., Call by Value/Ref.)		1	5 min
5 (Array, Pointer, C-String)		1	5 min
6 (Rekursion)		1	5 min
7 (Vererbung)		1	5 min
8 (Fehlertypen)		3	5 min
9 (Fehlersuche)		3	10 min
Zusatzaufgabe		0 (+2)	10 min
Summe		27	80 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, etc. dürfen nicht verwendet werden. Mobiltelefone bitte ausschalten (oder mit Aufsicht besprechen).
- Die Klausur hat 15 Seiten. Bitte prüfen Sie die Vollständigkeit.
- 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. Notfalls rufen Sie eine Aufsichtsperson zur Kontrolle.
- Bei Aufgabe 8 zum Ankreuzen sollten Sie wenigstens raten, wenn Sie die richtige Lösung nicht wissen (wenn Sie nichts ankreuzen, haben Sie den Punkt auf jeden Fall verloren). Es ist jeweils genau eine Antwort pro Teilaufgabe richtig.
- 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 Integer-Division ist für positive Eingabewerte garantiert, daß abgerundet wird, z.B. liefert `8/3` das Ergebnis `2`.

Aufgabe 1 (Programmierung: Klasse) 9 Punkte

Programmieren Sie eine Klasse `ware`. Objekte der Klasse `ware` sollen zwei Attribute haben: `name` und `preis`.

- Der Name soll ein klassischer C-String sein, deklarieren Sie dieses Attribut also als `const char *`. Sie brauchen die Zeichenkette nicht zu kopieren, sondern sich im Objekt nur den Zeiger merken.
- Der Preis soll ein `int`-Wert sein (Preis in Cent).

Ihre Klasse soll drei Methoden haben:

- Den Konstruktor mit Parametern für die beiden Attribute. Bei der Konstruktion eines Objektes sollen also Name und Preis mit angegeben werden, z.B.

  ```
  ware w1("Milch", 50);
  ```

  (wenn Milch 50 Cent kostet). Der Konstruktor soll testen, ob der gegebene Preis nicht negativ ist (also größer oder gleich 0), andernfalls soll das die Methode `fehler` der Klasse `fehlerbehandlung` mit der Meldung `"Preis nicht positiv"` aufgerufen werden:

  ```
  class fehlerbehandlung {
  public:
      static void fehler(const char *meldung);
  }
  ```

  Auch im Fehlerfall sollen die Attribute des Objektes korrekt initialisiert werden, der Preis dann auf 0.
- Die Methode `get_name`, die den Namen liefert. Sie hat natürlich keine Parameter.
- Die Methode `get_preis`, die den Preis liefert (ebenfalls ohne Parameter).

Die Attribute sollen von außen nur über die Methoden `get_name` und `get_preis` zugreifbar sein, man soll z.B. außerhalb der Klasse nicht direkt eine Zuweisung schreiben können. Da die Klasse über keine Änderungsfunktionen verfügt, bleiben Name und Preis nach der Konstruktion fest (es hängt von der jeweiligen Anwendung ab, ob das realistisch ist). Dagegen sollen die drei Methoden selbstverständlich von außen zugreifbar sein.

In der zweiten Aufgabe finden Sie eine Anwendung der Klasse `ware` und ein kurzes Testprogramm. Falls Ihnen die Aufgabenstellung nicht ganz klar ist, könnte es eventuell helfen, sich das schon anzuschauen.

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 für die Losung von Aufgabe 1 (Klasse `ware`):

Aufgabe 2 (Programmierung: Array) 6 Punkte

Gegeben sei folgende Klasse für Kassenbons/Kassenzettel (Quittungen z.B. von Lebensmittelgeschäften). Sie enthält eine Folge von Zeigern auf Objekte der Klasse `ware` in einem Array. Immer wenn eine Ware eingescannt oder eingetippt wird, wird ein Zeiger auf das entsprechende `ware`-Objekt im Array an die nächste freie Position gespeichert. Das Array kann also mehrfach den gleichen Wert (Zeiger auf das gleiche `ware`-Objekt) enthalten, nicht unbedingt hintereinander (mehrere Stück der gleichen Ware müssen auf dem Band nicht immer zusammen liegen).

```
typedef ware *ware_t; // Wir arbeiten mit Zeigern auf ware-Objekte

class bon {
private:
    static const int MAX_WAREN = 100;
    ware_t waren[MAX_WAREN];
    ware_t *naechster_eintrag; // naechster freier Eintrag im Array
    int noch_frei;             // Anzahl noch freier Positionen

public:
    // Konstruktor:
    bon()
    {
        naechster_eintrag = waren;
        noch_frei = MAX_WAREN;
    };

    // Methode zum Anhaengen einer Ware an den Bon:
    void gescannt(ware_t w)
    {
        if(noch_frei == 0)
            cerr << "zu viele Waren auf einem Bon";
        else {
            *naechster_eintrag++ = w;
            noch_frei--;
        }
    };

    // Hier kommt Ihre Methode:
    int rabatt(ware_t w, int n)
    ...
};

const int bon::MAX_WAREN;
```

Der Laden hat öfters Sonderangebote der Form “Kaufen Sie 5 Mehrkornbrötchen zum Preis von 4”, allgemein: Wenn man `n` Stück der gleichen Ware `w` kauft, muß man nur

`n-1` bezahlen, d.h. man bekommt einen Rabatt in Höhe des Preises der Ware `w`.

Man kann das Sonderangebot auch mehrfach in Anspruch nehmen, wenn man im obigen Beispiel etwa 10 Mehrkornbrötchen kauft, muß man nur 8 bezahlen, d.h. der Rabatt ist dann `2*` der Preis eines Mehrkornbrötchens. Kauft man dagegen 9 Stück, ist der Rabatt nur der einfache Preis. Selbstverständlich kann der Rabatt auch `0` sein, wenn man weniger als `n` Stück der Ware `w` gekauft hat.

Schreiben Sie die Methode `rabatt` mit den Parametern `w` und `n`, wie oben angegeben. Sie müssen dazu die Methode `get_preis` der Klasse `ware` verwenden: Sie hat keine Parameter und liefert einen `int`-Wert (Preis in Cent). Ihre Methode soll ja den Rabatt in Cent liefert, nicht den Rabatt in Stück.

Implementierungshinweis: Sie müssen die im Array eingetragenen Waren durchlaufen, und mitzählen, wie häufig die Ware `w` vorkommt. Natürlich dürfen Sie nur den initialisierten Teil des Arrays abfragen (siehe Attribute `naechster_eintrag` bzw. `noch_frei`).

Noch ein Hinweis: Der Vergleich der Einträge im Array und der gegebenen Ware `w` soll soll die Adresse der Objekt vergleichen (also Werte vom Typ `ware_t`). Verschiedene Objekte zählen als verschiedene Waren (selbst wenn sie zufällig den gleichen Namen hätten — das wird die Anwendung verhindern).

Und noch ein Tipp zur Fehler-Vermeidung: Beachten Sie beim Aufruf von `get_preis`, das Werte vom Typ `ware_t` Zeiger auf Objekte sind.

Hier ist noch ein Test-Programm, um die Aufgabenstellung zu verdeutlichen:

```
... // Deklarationen von ware, ware_t, bon

int main () {
    ware_t milch  = new ware("milch", 50);
    ware_t butter = new ware("butter",80);
    bon mein_kassenzettel;

    mein_kassenzettel.gescannt(butter);
    mein_kassenzettel.gescannt(butter);
    mein_kassenzettel.gescannt(milch);
    for (int i=1; i <= 5; i++)
        mein_kassenzettel.gescannt(butter);
    cout << "Aktion: Kauf 3 zahle 2 (fuer Butter)!\n";
    cout << "Rabatt beim Kauf von 7 Stck Butter: " <<
            mein_kassenzettel.rabatt(butter, 3) << " Cent \n";

    return 0;
}
```

Die Ausgabe dieses Programmstücks ist:

```
Aktion: Kauf 3 zahle 2 (fuer Butter)!
Rabatt beim Kauf von 7 Stck Butter: 160 Cent
```

Platz für die Lösung:

```
int rabatt(ware_t w, int n)
```


Aufgabe 3 (Arithmetischer Ausdruck, Toter Code) 2 Punkte

Betrachten Sie folgende Funktion:

```
int f(int n, int m)
{
    int i, j;
    i = n + 1;
    i++;
    j = 27 * m;
    n = j % 5;
    j = i * m;
    if(j < 0 && j > 100)
        j = 0;
    return j;
}
```

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:

```
int f(int n, int m)

{

    return ________________________________________;

}
```

Aufgabe 4 (Globale/lokale Var., Call by Value/Ref.) 1 Punkt

Was gibt dieses Programm aus?

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

int n = 5;

int f(int i)
{
    int n = 3;
    g(&n, i);
    i = n;
    return i;
}

void g(int *a, int b)
{
    *a = (*a) * n * b;
}


int main()
{
    int n = 2;
    int j  = f(n);
    cout << n * 100 + j;

    return 0;
}
```

Ausgabe: ______________________________

Aufgabe 5 (Array, Pointer, C-String) 1 Punkt

Was gibt dieses Programm aus?

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

int main()
{
    char a[10];
    char *p = a;
    const char *q = "xyz";
    while(*q)
        *p++ = *q++;
    cout << a[1];

    return 0;
}
```

Ausgabe: ______________________________

Aufgabe 6 (Rekursion) 1 Punkt

Was gibt dieses Programm aus?

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

int n = 0;

void f(int i)
{
    if(i > 5) {
        n++;
        f(i-1);
    }
    else
        cout << i << '+' << n;
}

int main()
{
    f(8);

    return 0;
}
```

Ausgabe: ______________________________

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 10; }
};

class D : public C {
    public:
        int g() { return 5; }
        int h() { return 20; }
};

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 würde der Compiler bemerken und eine Fehlermeldung ausgeben? (Gehen Sie dabei von einem normalen Compiler aus, ohne einen extrem hohen Warnungslevel: Es zählen also nur echte Fehler, keine Warnungen.) Wenn Sie nichts ankreuzen, ist der Punkt auf jeden Fall verloren. Sie sollten daher notfalls raten.

a) Variable nicht deklariert.

- ☐ Der Compiler meldet den Fehler.
- ☐ Dieser Fehler wird vom Compiler nicht gemeldet und kann nur durch Testen gefunden werden.

b) Endlosschleife.

- ☐ Der Compiler meldet den Fehler.
- ☐ Dieser Fehler wird vom Compiler nicht gemeldet und kann nur durch Testen gefunden werden.

c) Übergabe eines `int`-Wertes (z.B. 5) für einen Funktionsparameter, der als `int *p` deklariert ist.

- ☐ Der Compiler meldet den Fehler.
- ☐ Dieser Fehler wird vom Compiler nicht gemeldet und kann nur durch Testen gefunden werden.

Aufgabe 9 (Fehlersuche) 3 Punkte

Das folgende Programm enthält (mindestens) 5 Fehler. Bitte geben Sie drei dieser Fehler an (es ist möglicherweise schwierig, alle fünf 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 f() { return a * a; }
 (8)  public:
 (9)          int g(int i) { return f() + i; }
(10)          C(int n) { a = n; }
(11)  };
(12)
(13)  int main()
(14)  {
(15)          C* x = new C(5);
(16)          int i = x->f();
(17)          (i+1) = 4;
(18)          i += g(27);
(19)          if(i = 0) i++;
(20)          cout << i + x.g(3);
(21)          return 0;
(22)  }
```

1. Zeile: ______

   Begründung: ____

   ____

2. Zeile: ______

   Begründung: ____

   ____

3. Zeile: ______

   Begründung: ____

   ____

Zusatzaufgabe (Programmierung) 2 Extrapunkte

Erweitern Sie die Klasse `ware` aus Aufgabe 1 um eine Methode `print_preis` (ohne Parameter). Diese Methode soll den Preis in der Form "Euro.Cent" auf `cout` ausgeben. Sie müssen also den abgespeicherten Preis (in Cent) durch 100 teilen und das Ergebnis ausgeben, dann einen Punkt, dann eventuell eine zusätzliche 0 (falls der Cent-Wert zwischen 0 und 9 liegt), und dann den Cent-Wert (Rest bei Division des gespeicherten Preises durch 100). Wenn der gespeicherte Preis also z.B. `109` ist, muß diese Methode `1.09` ausgeben.

Sie brauchen nur die Methode selbst zu schreiben (nicht noch einmal die Klasse oder eine Subklasse etc.).

Platz für die Lösung: