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

15. Januar 2006

Vorlesung "Objektorientierte Programmierung" — Musterlösung zu Probeklausur II —

Die Probeklausur wurde in den Übungsgruppen geschrieben. Es gab insgesamt 61 Teilnehmer. Die Klausur ist schlecht ausgefallen. Es gab 10 Punkte, davon wurden erreicht:

Punkte	Anz. Teilnehmer	Note ca.
9.0	5	1.0 oder 1.3
8.5	1	1.3
8.0	7	1.7 oder 2.0
7.0	4	2.3 oder 2.7
6.5	1	3.0 oder 3.3
6.0	7	3.3 oder 3.7
5.0	3	4.0 oder 5.0
4.0	8	5.0
3.5	2	5.0
3.0	7	5.0
2.0	7	5.0
1.5	1	5.0
1.0	1	5.0
0.5	2	5.0
0.0	5	5.0

Die Angaben über Zensuren sollen nur einen ungefähren Anhaltspunkt geben. Es gibt keine Garantie, daß die Endklausur genauso bewertet wird. Selbst wenn die Grenze bei 50% der Punkte gesetzt wird (garantiert bestanden haben Sie nur mit 60%), wären 33 Teilnehmer, also 54% durchgefallen.

Aufgabe 1 (Rekursive Funktion) 1 Punkt

Was gibt das folgende Programm aus?

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

void f(int n)
{
        if(n > 0) {
                cout << -n << ' ';
                f(n - 1);
                cout <<  n << ' ';
        }
        else
                cout << "* ";
}

int main()
{
        f(2);

        return 0;
}
```

Lösung:

Das Programm gibt

`-2 -1 * 1 2`

aus.

Die Funktion `f` wird erst mit dem Parameterwert `n = 2` ausgerufen. Die Bedingung `n > 0` ist also erfüllt, daher druckt sie `-2` aus (gefolgt von einem Leerzeichen), und ruft sich dann selbst mit dem Parameterwert `n = 1` auf.

Auch in diesem Fall gilt `n > 0`, also wird `-1` gedruckt, und es kommt zum rekursiven Aufruf mit `n = 0`.

Hier ist die `if`-Bedingung nicht erfüllt, also wird der `else`-Zweig ausgeführt, und das Sternchen (`*`) gedruckt (wieder gefolgt von einem Leerzeichen zur Trennung).

Dann kehrt der rekursive Aufruf für `n = 0` zurück zum Prozeduraufruf für `n = 1`. Hier steht ja nach dem Aufruf von `f` noch ein weiterer Befehl, nämlich `n` auszugeben. Jetzt wird also `1` gedruckt. Es war ein häufiger Fehler, diese Rückkehr zu vergessen.

Anschließend kehrt auch dieser rekursive Aufruf zurück zum Aufrufer, nämlich der Prozeduraktivierung für `n = 2`. Es wird dann entsprechend `2` ausgegeben.

Nun kehrt dieser Aufruf von `f` zurück zu `main`, aber das Hauptprogramm beendet sich dann auch. Es signalisiert mit `return 0`, daß das Programm erfolgreich ausgeführt wurde.

Punkte	Teilnehmer	Prozent
1	25	41%
0	36	59%

Aufgabe 2 (Lokale Variablen, Parameterübergabe) 1 Punkt

Was gibt das folgende Programm aus?

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

int n = 1;

int f(int n)
{
        n++;
        n = n * 2;
        if(n > 20)
                n = n << 3;
        n *= 0;
        return n;
}

void g(int *p)
{
        (*p)--;
        n += 5;
        cout << n << ' ';
}

int main()
{
        int n = 4;
        int k = f(n);
        g(&n);
        cout << n;
        return k;
}
```

Lösung:

Das Programm druckt

```
6 3
```

aus. Zunächst wird in `main` eine lokale Variable `n` auf `4` gesetzt. Dieser Wert wird dann an `f` übergeben. Die Parameterübergabe ist aber "call by value", d.h. `f` bekommt nur den Wert `4`, aber keinen Zugriff auf die Variable `n`. Daher braucht man sich all die komplizierten Sachen, die `f` mit `n` macht, gar nicht anzuschauen. Der Rückgabewert von `f` wird ja nicht ausgegeben. Übrigens wird der Wert von `n` in `f` kurz vor Schluß mit `0` multipliziert (`n *= 0` bedeutet `n = n * 0`), daher gibt `f` den Wert `0` zurück. Das spielt aber für die Aufgabe keine Rolle. Das `n` in `main` hat noch immer den Wert `4`. Jetzt wird aber die Adresse von `n` an `g` übergeben. Daher erhält `g` Zugriff auf die Variable, die uns interessiert. In `g` spricht man diese Variable mit `*p` an Daher bewirkt `(*p)--`, daß `n` (die lokale Variable aus `main`). auf `3` verringert wird. Bevor `g` zurückkehrt, erhöht sie aber noch die gloable Variable `n` um `5` und druckt sie anschließend aus. An dieser Stelle wird also `6` ausgegeben. Dann kehrt `g` zu `main` zurück, hier wird die lokale Variable `n` ausgegeben, also `3`.

Punkte	Teilnehmer	Prozent
1	8	13%
0	53	87%

Aufgabe 3 (Strings, Pointer) 1 Punkt

Was gibt dieses Programm aus?

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

int main()
{
        const char *p = "123";
        char a[10];
        char *q = a;
        while(*p) {
                *q = *p;
                p++;
                q++;
        }
        a[1] = '\0';
        cout << a;
        return 0;
}
```

Lösung:

Das Programm druckt

```
1
```

aus. Zunächst zeigt der Zeiger `p` auf die Zeichenkette `"123"` und der Zeiger `q` auf das Zeichenarray `a`.

Die `while`-Schleife ist eine typische Kopierschleife für Zeichenketten (man würde es aber normalerweise mit `*q++ = *p++;` noch etwas kompakter aufschreiben): Solange `p` nicht auf das Nullbyte zeigt, das den String abschließt, wird das Zeichen, auf das `p` zeigt, in die Speicherstelle kopiert, auf die `q` zeigt. Anschließend werden beide Zeiger erhöht. Danach stehen in dem Array `a` also folgende Zeichen:

- `a[0] = '1'`,
- `a[1] = '2'`,
- `a[2] = '3'`.

Nun wird `a[1]` aber mit dem Nullbyte überschrieben, das das Ende von Zeichenketten markiert. Wenn man `a` anschließend ausdruckt, wird also nur `a[0]` ausgegeben, und das ist das Zeichen `'1'`.

Punkte	Teilnehmer	Prozent
1	27	44%
0	34	56%

Aufgabe 4 (Programm) 5 Punkte

Schreiben Sie ein Programm, das 10 ganze Zahlen (`int`-Werte) einliest, und dann "`ja`" ausgibt, falls eine Zahl in der Folge doppelt vorgekommen ist, und "`nein`" sonst. Z.B. könnte die Eingabe sein:

```
3 45 7 -6 98 4 55 0 1 2
```

In diesem Fall soll "`nein`" ausgegeben werden, da keine Zahl doppelt vorkommt. Bei der Eingabe

```
0 1 2 3 4 2 5 6 7 8
```

soll dagegen "`ja`" ausgegeben werden, weil die `2` doppelt ist. Die Ausgabe soll erst erfolgen, nachdem alle Zahlen eingelesen wurden. Fehler bei der Eingabe brauchen Sie nicht zu behandeln. Es soll leicht möglich sein, die Zahl 10 durch eine andere Zahl zu ersetzen. Bemühen Sie sich um Verständlichkeit und guten Programmierstil (es gibt ggf. Punktabzüge). Schreiben Sie das Programm vollständig und ohne Syntaxfehler.

Lösung:

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

int main ()
{
   const int laenge = 10;

   int feld[laenge];

   for (int i=0; i < laenge; i++)
      cin >> feld[i];

   bool gleich = false;

   for (int i=0; (i < laenge) && (! gleich); i++)
      for (int k=i+1; (k < laenge) && (! gleich); k++)
         if ( feld[i] == feld[k] ) gleich = true;

   if (gleich) cout << "ja";
   else        cout << "nein";

   return 0;
}
```

Punkte	Teilnehmer	Prozent
5	10	16%
4	12	20%
3	15	25%
2	7	11%
1	6	10%
0	11	18%

Aufgabe 5 (Fehlersuche, Strukturen) 2 Punkte

Das folgende Programm enthält (mindestens) 3 Fehler, die der Compiler finden sollte. Bitte geben Sie zwei dieser Fehler an (es ist möglicherweise schwierig, alle drei zu entdecken). Sie bekommen keinen Extrapunkt, wenn Sie alle drei finden. Falls Sie mehr als zwei Fehler angeben, werden nur die ersten beiden gewertet. Geben Sie bitte die Zeilennummer mit an, in der der Compiler den Fehler finden müßte, wenn er das Programm von oben nach unten liest.

```
 (1)  #include <iostream>
 (2)  using namespace std;
 (3)
 (4)  struct s1 { int i; };
 (5)  struct s2 { int i; };
 (6)
 (7)  int main()
 (8)  {
 (9)          int i;
(10)          s1 x1;
(11)          s2 x2;
(12)          cin >> i;
(13)          x1.i = i;
(14)          x2 = x1;
(15)          int *p = &x2;
(16)          if(x1->i == 0)
(17)                  cout << 1;
(18)          else
(19)                  cout << 0;
(20)          return 0;
(21)  }
```

Lösung:

- `x2 = x1;` (Zeile 14)
  Die Zuweisung ist nicht möglich, weil Strukturen mit unterschiedlichem Namen in C++ als unterschiedliche Typen gelten. Es spielt dann keine Rolle mehr, daß die Strukturen tatsächlich gleich aufgebaut sind. Zuweisungen zwischen unterschiedlichen Typen sind in C++ grundsätzlich nicht erlaubt, obwohl es viele implizite Typumwandlungen gibt, die zwischen bestimmten Typen dann doch eine Zuweisung erlauben. Für verschiedene Struktur-Typen gibt es aber keine solche Umwandlung, daher ist die Zuweisung ein Fehler.

- `int *p = &x2;` (Zeile 15)
  Auch hier passen die Typen nicht. Die rechte Seite liefert einen Zeiger auf eine Struktur, die linke Seite braucht einen Zeiger auf ein Integer. Es spielt dabei keine Rolle, daß die Struktur tatsächlich nur aus einem einzigen Integer besteht. Die Struktur ist ein neuer Typ, Zuweisungen zwischen verschiedenen Zeigertypen sind nicht erlaubt (außer bei `void*` auf der linken Seite).

- `x1->i` (Zeile 16)
  `x1` ist eine Struktur, und kein Zeiger auf eine Struktur. Deswegen muß es richtig `x1.i` heißen. `x1->i` ist eine Abkürzung für `(*x1).i`. Man kann aber `x1` nicht dereferenzieren, weil es kein Zeiger ist.

Punkte	Teilnehmer	Prozent
2	16	26%
1.5	3	5%
1	17	28%
0.5	4	7%
0	21	34%

Umfrage

- Wieviel Prozent des Vorlesungsstoffes sind für Sie neu?

Prozent neu	Teilnehmer
0%	2
25%	11
50%	14
75%	17
100%	16

(Ein Teilnehmer hat an der Umfrage nicht teilgenommen.)

- Gesamtpunktzahl bei 0% oder 25% neu:

Punkte	Teilnehmer
9	2
8	3
6.5	1
6	3
5	2
4	1
2	1

- Gesamtpunktzahl bei 100% neu:

Punkte	Teilnehmer
8.5	1
7	1
6	2
4	1
3.5	2
3	2
2	3
1.5	1
0	3

- Gesamtpunktzahl bei 75% neu:

Punkte	Teilnehmer
9	2
8	1
7	1
6	1
4	4
3	2
2	3
1	1
0.5	1
0	1

- Besuchen Sie regelmäßig die Vorlesung?

Regelmäßiger Besuch	Teilnehmer
fast immer	45
öfters	12
fast nie	3