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

01. Februar 2006

Vorlesung "Objektorientierte Programmierung" — Musterlösung zur Probeklausur III —

Aufgabe 1 (Rekursive Funktion) 1 Punkt

Was gibt das folgende Programm bei der Eingabe `a = 4` und `b = 2` aus? Es empfihlt sich, zur Lösung der Aufgabe einen Rekursionsbaum zu zeichnen (welcher Funktionsaufruf erzeugt welche anderen Funktionsaufrufe, wobei die Funktionsaufrufe durch die aktuellen Eingabeparameter dargestellt sind).

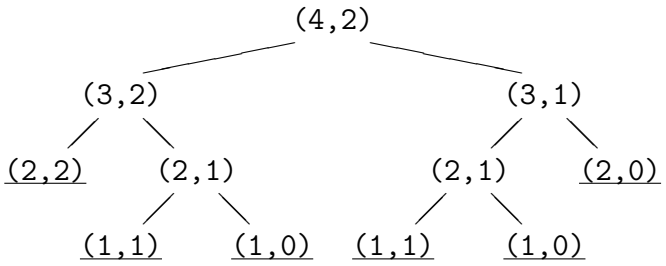
```
#include <iostream>
using namespace std;

unsigned long f(unsigned long a, unsigned long b);

int main()
{
  unsigned long a, b;

  cout << "Bitte geben Sie a ein: ";
  cin >> a;

  cout << "Bitte geben Sie b ein: ";
  cin >> b;

  cout << "Ergebnis = " << f(a, b) << endl;

  return 0;
}

unsigned long f(unsigned long a, unsigned long b)
{
  if (a == b || b == 0)
    return 1L;

  if (a < b)
    return 0L;

  return f(a - 1, b) + f(a - 1, b - 1);
}
```

Lösung:

Für die Eingaben `a=4, b=2` gibt das Programm

```
6
```

aus. Man zeichnet sich am besten einen Baum der rekursiven Aufrufe auf, wobei die Knoten mit den Werten für die EIngabeparameter `(a,b)` markiert sind. Der Fall `a < b` tritt nicht auf, die Aufrufe stoppen schon vorher mit `a == b`. Daher werden am Ende einfach die Blätter des Rekursionsbaums gezählt (für jedes Blatt wird 1 aufaddiert). Im Bild sind die Blätter unterstrichen:



Aufgabe 2 (Verständnisfragen) 4 Punkte

Beantworten Sie bitte folgende Fragen:

1. Was ist der Unterschied zwischen einem Feld (Array) und einem Zeiger?

   Bei einem Feld ist Speicherplatz reserviert, auf den der Zeiger zeigt. Bei einem Zeiger muß man selbst eine Adresse zuweisen. Dafür kann man den Zeiger ändern, das Feld ist ein konstanter Zeiger.

2. Wie ist der Zusammenhang zwischen Zeigern und Referenzen?

   Eine Referenz ist intern auch ein Zeiger (wenn der Compiler nichts optimiert hat), aber sie wird immer automatisch dereferenziert. Bei einem Zeiger muß man dagegen explizit den Dereferenzeirungsoperator `*` verwenden.

3. In welcher Situation braucht man einen Kopierkonstruktor?

   Bei der Übergabe eines Objektes “by value” an eine Funktion. Außerdem auch bei der Rückgabe eines Objektes als Funktionswert.

4. Wie viele Rückgabewerte kann eine Funktion haben?

   Einen. Falls man mehr Werte zurückgeben will, muß man Parameter von einem Zeiger- oder Referenztyp verwenden.

Aufgabe 3 (Vererbung) 1 Punkt

Was gibt das folgende Programm aus?

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

class A {
    public:
        int f() { return 1; }
        virtual int g() { return 2; }
};

class B : public A {
    public:
        int f() { return 3; }
        int g() { return 4; }
};

int main()
{
        A a;
        B b;
        A* p1 = &a;
        A* p2 = &b;
        B* p3 = &b;

        cout << "p1: " << p1-> f() << ' ' << p1->g() << '\n';
        cout << "p2: " << p2-> f() << ' ' << p2->g() << '\n';
        cout << "p3: " << p3-> f() << ' ' << p3->g() << '\n';

        return 0;
}
```

Lösung:

Das Programm gibt Folgendes aus:

```
p1: 1 2
p2: 1 4
p3: 3 4
```

- `p1` ist ein Zeiger auf ein Objekt der Klasse `A`. Deshalb werden natürlich die dort definierten Funktionen benutzt.
- `p2` ist ein Zeiger auf ein Objekt der Klasse `B`, das aber hier als Objekt der Klasse `A` behandelt wird (der Zeiger ist vom Typ `A*`). Daher wird für die Funktion `f` die in `A` definierte Version aufgerufen. Für die Funktion `g` wird dagegen die in `B` definierte Version aufgerufen, da `g` als `virtual` deklariert ist, also das Objekt selbst einen Zeiger auf die Funktion enthält (über die Virtual Function Table), insofern bringt das Objekt seine Funktion selbst mit.
- Für `p3` ist die Situation wieder klar: Es ist ein Objekt der Klasse `B`, das auch über einen Zeiger passenden Typs angesprochen wird. Also werden die in `B` definierten Funktionen verwendet.

Aufgabe 4 (Programm) 8 Punkte

Definieren Sie eine Klasse `Strecke` zur Darstellung einer Strecke im zweidimensionalen kartesischen Koordinatensystem. Diese besitzt die folgenden vier `private`-Datenelemente: `start_x, start_y, end_x, end_y` vom Typ `double`.

Deklarieren und definieren Sie folgende drei Methoden als `public`:

- `set_start`:
  Diese Methode soll den den Startpunkt (`start_x, start_y`) der Strecke initialisieren. Die Werte `x` und `y` für den Startpunkt werden als Parameter übergeben.
- `set_end`:
  Diese Methode soll entsprechend den Endpunkt (`end_x, end_y`) der Strecke initialisieren. Die Werte `x` und `y` für den Endpunkt werden als Parameter übergeben.
- `get_laenge`:
  Diese Methode soll die Länge der Strecke liefern (als Returnwert vom Typ `double`). Die Berechnung der Länge erfolgt über den Satz des Pythagoras, d.h.

  $$\sqrt{(\texttt{end\_x} - \texttt{start\_x})^2 \; + \; (\texttt{end\_y} - \texttt{start\_y})^2}$$

  Verwenden Sie zur Berechnung der Quadratwurzel die Funktion

  ```
  double sqrt(double x).
  ```

  Diese Funktion wird in der Header-Datei `math.h` deklariert.

Bemühen Sie sich um Verständlichkeit und guten Programmierstil (es können auch Punkte für schlechten Stil abgezogen werden).

Schreiben Sie die Klassendefinition vollständig und ohne Syntaxfehler. Sie brauchen keinen Konstruktor und Destruktor zu definieren. Schreiben Sie bitte nur die Klassendefinition, kein Hauptprogramm.

Lösung:

```
#include <math.h>

class Strecke {
    private:
        double start_x;
        double start_y;
        double end_x;
        double end_y;
    public:
        void set_start(double x, double y)
        {
                start_x = x;
                start_y = y;
        }
        void set_end(double x, double y)
        {
                end_x = x;
                end_y = y;
        }
        double get_laenge()
        {
                double diff_x = end_x - start_x;
                double diff_y = end_y - start_y;
                return sqrt(diff_x*diff_x + diff_y*diff_y);
        }
};
```

Aufgabe 5 (Fehlersuche, Speicherverwaltung) 2 Punkte

Das folgende Programm enthält drei Fehler im Umgang mit Speicher. Diese Fehler werden durch den Compiler nicht unbedingt erkannt, können dann aber zur Laufzeit des Programmes zum Absturz oder anderem Fehlverhalten führen.

Bitte geben Sie zwei dieser Fehler an (es ist möglicherweise schwierig, alle drei zu entdecken). Falls Sie mehr als zwei Fehler angeben, werden nur die ersten beiden gewertet.

```
 1: #include <iostream>
 2: using namespace std;
 3:
 4: const int LEN=32;
 5: const int ITER=1000000;
 6: typedef struct Bell
 7: {
 8:     char *ding;
 9:     char *dong;
10: };
11:
12: char* f1 (void)
13: {
14:    char carray[LEN];
15:    for(int i = 0; i < LEN; i++)
16:       carray[i] = 'a';
17:    carray[LEN] = '\0';
18:    return carray;
19: }
20:
21: char* f2()
22: {
23:   return new char[1024];
24: }
25:
26: int main (void)
27: {
28:    Bell *f;
29:
30:    for(int i = 0; i < ITER; i++) {
31:       f = new Bell;
32:       f->ding = f1();
33:       f->ding[0] = 'b';
34:       f->dong = f2();
35:       f->dong[0] = 'c';
36:       f->dong[1] = '\0';
37:       cout << f->ding << f->dong << '\n';
38:       delete f;
39:    }
40:    return 0;
41: }
```

- Zeile 17:
  Hier wird auf das Array außerhalb seiner Grenzen zugegriffen. Der Indexbereich geht nur bis `LEN-1`.
- Zeile 18:
  Hier wird die lokale Variable `carray` als Funktionswert zurückgeliefert. Das ist aber ein Zeiger auf eine Speicherstelle, die bei der Rückkehr der Funktion freigegeben wird. Damit wird die Zuweisung an diese Speicherstelle in Zeile 33 gefährlich (vermutlich stehen dann an der Stelle auf dem Stack gerade keine wichtigen Daten, so daß nichts passiert, aber es ist dennoch eindeutig ein Fehler).
- Zeile 38:
  Hier wird zwar die Struktur `f` wieder freigegeben, aber nicht das Zeichenarray, auf das `dong` zeigt, und das von `f2` dynamisch angefordert wurde. Dies ist ein Fall von einem Speicherleck. Das ist besonders ungünstig, weil es in einer Schleife geschieht. Wenn das Programm bis zu Ende durchläuft, würde es 1 GB an Speicher aufgrund dieses Speicherlecks belegen.