ÜBUNG 8.1 OBJEKTVERHALTEN IN C++

```
5  class Zahl {
6  public:
7    Zahl();                  // Konstruktor
8    Zahl( const int& n );    // int-Konstruktor
9    Zahl( const Zahl& n );   // Copy-Konstruktor
10   ~Zahl();                 // Destruktor
11   Zahl& operator=( const int& n );
12     // int-Zuweisung
13   Zahl& operator=( const Zahl& n );
14     // Zuweisung
15   Zahl operator+( const Zahl& n );
16     // Addition
17 private:
18   int z;
19 };
20
21 Zahl::Zahl() { z = 0; }
22 Zahl::Zahl( const int& n )  { z = n; }
23 Zahl::Zahl( const Zahl& n ) { z = n.z; }
24 Zahl::~Zahl()               { }
25 Zahl& Zahl::operator=( const int& n )
26     { z=n;    return *this; }
27 Zahl& Zahl::operator=( const Zahl& n )
28     { z=n.z; return *this; }
29 Zahl Zahl::operator+( const Zahl& n )
30     { return Zahl(z+n.z); }
31
32 Zahl f( Zahl a, Zahl& b) { return a+b; }
33
34 int main (void)
35 {
36   Zahl a(3), b(4);
37   Zahl c(a), d, e;
38   d = 5;
39   d = f(d,c);
40   e = d + a + b;
41   return 0;
42 }
```

Die Klasse `Zahl` ist eine (unvollständige) Implementierung einer Klasse ganzer Zahlen. Geben Sie an, welche Methoden der Klasse `Zahl` bei der Ausführung des obigen Programmes in welcher Reihenfolge aufgerufen werden. Geben Sie dazu für die Zeilen 36–41 an, welche Methoden für welches Objekt aufgerufen werden. Wenn in einer Zeile n der Konstruktor für ein Objekt u, der Zuweisungsoperator für ein Objekt v und der Destruktor für u aufgerufen werden, dann schreiben Sie:

$$n: \qquad u \text{ Konstuktor}, \quad v \text{ Zuweisung}, \quad u \text{ Destruktor}$$

Natürlich sollen Sie auch *temporäre Objekte** erwähnen, die z.B. bei der Addition lokal innerhalb der Methode erzeugt werden. Temporäre Objekte bezeichnen Sie mit Großbuchstaben.
Beispiel: Der Ausdruck $x = y + z$ erzeugt folgende Aufrufe:

1. Die Additionsmethode von y.
2. Der `int`-Konstruktor in der Additionsmethode erzeugt ein neues Objekt T.
3. Der Copy-Konstruktor erzeugt ein neues Objekt U zur Rückgabe des in der Additionsmethode erzeugten Objekts (denn Rückgabetyp ist `Zahl` und nicht `Zahl&` – eine Referenz zurückzuliefern scheint eine Möglichkeit diese zusätzliche Kopie zu umgehen, wäre aber verhängnisvoll, denn das wäre eine Referenz auf eine lokale temporäre Variable, die bald zerstört wird.)
4. Destruktor des Objektes T.
5. Zuweisungsoperator von x.
6. Destruktor von U.

Angenommen, der Ausdruck $x = y + z$ steht in Zeile 40. Dann sieht das Ganze in Kurzform – also in der Form, in der Sie die Lösung dieser Aufgabe abgeben sollen – folgendermaßen aus:

*In C++ werden temporäre Objekte erzeugt, wenn ein Objekt an eine Methode oder Funktion übergeben wird und wenn eine Methode oder Funktion ein Objekt zurückliefert. In diesem Fall wird nicht das Objekt selbst übergeben, sondern eine Kopie, es sei denn die Übergabe geschieht mit einer Referenz. In einem Ausdruck der Form $a+b+c$ wird zuerst $a+b$ ausgewertet (siehe auch `http://de.cppreference.com/w/cpp/language/operator_precedence`), indem der Additionsoperator für a mit dem Argument b aufgerufen wird. Das Ergebnis wird in einem temporären Objekt gespeichert,dessen Additionsmethode dann mit dem Argument c aufgerufen wird. Objekte werden in umgekehrter Reihenfolge ihrer Erzeugung zerstört.

40: y Addition T `int`-Konstruktor U Copy-Konstruktor T Destruktor x Zuweisung U Destruktor

Das Programm gibt es unter

`http://conan.iwr.uni-heidelberg.de/teaching/info1_ws2014/c++/defmet.cc`

Sie können sich von ihrem Compiler helfen lassen, indem Sie die Methodenaufrufe auf der Konsole ausgeben. Beachten Sie dabei jedoch, dass ihr Compiler unter Umständen übereifrig optimiert. Verwenden Sie die Compileroption `-fno-elide-constructors` und überprüfen Sie auf jeden Fall ob Ihre Ergebnisse mit Ihrem Wissen erklärbar sind. *5 Punkte*

ÜBUNG 8.2 LISTEN-KLASSE

In der Vorlesung haben Sie Funktionen und Datenstrukturen kennengelernt `int`-Zahlen in einer einfach verketteten Liste zu verwalten.

a) Die in der Vorlesung vorgestellte Version hat den Nachteil, dass Funktionen und Daten getrennt sind. Schreiben Sie deshalb, basierend auf den in `intliste.cc` implementierten Funktionen, eine Klasse für eine Integerliste. Die Klasse sollte dabei mindestens folgendes Interface erfüllen:

```
class IntList {
public:
  // Konstruktor, erzeugt eine leere Liste
  IntList();
  // Destruktor, loescht gesamten Listeninhalt
  ~IntList();
  // Gibt Anzahl der Elemente zurueck
  int getCount();
  // Gibt zurueck, ob die Liste leer ist
  bool isEmpty();
  // Gibt die Liste aus
  void print();
  // Fuegt die Zahl 'element' an der (beliebigen) Position 'position' ein
  void insert(int element, int position);
  // Loescht das Element an der Position 'position'
  void remove(int position);
  // Gibt den Wert des Elements an der Position 'position' zurueck
  int getElement(int position);
private:
  // ... (hier folgen private Member der Klasse)
};
```

Überlegen Sie sich, welche privaten Daten und Methoden Sie brauchen und welche Funktionen Konstruktor und Destruktor ausführen sollten. Anhand der Methodensignaturen erkennen Sie bereits, dass der Benutzer nur mit `int`s zu tun hat, Dinge wie die Erzeugung von Listenelementen und das Hantieren mit Pointern ist Aufgabe Ihrer Klasse! [*5 Punkte*]

b) Es gibt in C++ eine Faustregel namens "Rule of Three"[†]. Diese Regel besagt, dass die drei Methoden

- Destruktor
- Copy-Konstruktor
- Zuweisungsoperator

immer zusammen auftreten sollten. Implementiert man eine dieser Methoden explizit, sollte man auch die anderen bereitstellen. Die dieser Regel zugrundeliegende Annahme ist folgende: Ist man für eine dieser Methoden mit der impliziten Standardvariante des Compiler nicht zufrieden, trifft dies höchstwahrscheinlich auch auf die anderen beiden Methoden zu. Dies ist meistens dann der Fall, wenn man mit Pointern hantiert.

Implementieren Sie für Ihre Klasse nun also die beiden fehlenden Methoden. Achten Sie bei der Implementierung des

[†]seit `C++11` eigentlich "Rule of Five", siehe `http://en.wikipedia.org/wiki/Rule_of_three_%28C%2B%2B_programming%29`

– Copy-Konstruktors darauf, dass Sie eine *tiefe* Kopie[‡] Ihrer Liste erstellen. Bloßes Kopieren der Pointer auf die Listenelemente (wie es der Compiler in seiner Default-Variante tun würde) reicht nicht aus, Sie müssen jedes Listenelement neu erstellen!
– Zuweisungsoperators darauf,
  * dass Sie die bereits vorhandenen Daten einer existierenden Liste "aufräumen", bevor Sie ihr neue Daten zuweisen.
  * dass ein Objekt niemals "sich selbst" zugewiesen wird. Den Test, ob das aktuelle Objekt gleich einem als Referenz übergebenen anderen `other` ist, kann man relativ einfach mit `if (this != &other)` überprüfen.

[*5 Punkte*]

10 Punkte

ÜBUNG 8.3 FEHLERRECHNUNG

Zur Berechnung von Naturkonstanten (wie z.B. im Physikpraktikum) möchte man üblicherweise wissen, innerhalb welcher Fehlergrenzen das Ergebnis liegt, da die Messwerte ungenau sind. Meistens misst man einen Wert w mehrfach und kennt dann den Mittelwert $\bar{w}$ und eine Standardabweichung (absoluter Fehler) Δw, also z.B. 56.3 ± 0.1.

Bei einer Rechnung mit mehreren solchen fehlerbehafteten Variablen nimmt man nun an, daß die Messfehler voneinander unabhängig und normalverteilt sind, denn dann gilt das Gaußsche Fehlerfortpflanzungsgesetz. Mit zwei Spezialfällen dieses Gesetzes wollen wir uns nun beschäftigen: Bei einer Summe

$$s = a + b$$

mit fehlerbehafteten Werten a und b ist der absolute Fehler

$$\Delta s = \sqrt{(\Delta a)^2 + (\Delta b)^2}$$

Ist also z.B. $a = 10 \pm 2$ und $b = 5 \pm 1$ ist die Summe $s = 15 \pm \sqrt{5} (\approx 2.2)$.

Bei Produkten muß der relative Fehler $\frac{\Delta w}{w}$ verwendet werden. Der relative Fehler des Ausdrucks

$$p = a \cdot b$$

errechnet sich durch

$$\frac{\Delta p}{p} = \sqrt{\left(\frac{\Delta a}{a}\right)^2 + \left(\frac{\Delta b}{b}\right)^2}$$

Mit den obigen Werten von a und b ergibt sich also $\frac{\Delta p}{p} \approx 0.2828$ (relativer Fehler) bzw. der absolute Fehler zu $\Delta p \approx 0.2828 * p = 14.14$. Wir schreiben dies kurz als $p = 50 \pm 28.28\%$ bzw. $p = 50 \pm 14.14$.

Schreiben Sie eine Klasse `FehlerWert` mit den Methoden `operator+` und `operator*`, so dass das folgende Hauptprogramm läuft:

```
int main () {
  // Konstruktor nimmt zwei double-Werte: den Wert und den absoluten Fehler
  FehlerWert a(10.0, 2.0), b(5.0, 1.0);
  FehlerWert s = a + b;
  std::cout << s.wert() << " +- " << s.absolut() <<
    " (" << s.relativ() * 100 << " %)" << std::endl;
  FehlerWert p = a * b;
  std::cout << p.wert() << " +- " << p.absolut() <<
    " (" << p.relativ() * 100 << " %)" << std::endl;
}
```

Mit dieser Klasse können Sie dann auch komplexere Formeln mit fehlerbehafteten Werten leicht berechnen.

5 Punkte

[‡] siehe `http://en.wikipedia.org/wiki/Object_copy`