Virtuelle Basisklassen

Falls mindestens zwei Klassen (**`class segelb, class motorb`**) von einer gemeinsamen Basisklasse (**`class boote`**) abgeleitet sind und eine weitere Klasse (**`class segelbmitmotor`**) von den beiden erstgenannten Klassen abgeleitet ist, dann existieren in der am weitesten abgeleiteten Klasse (**`class segelbmitmotor`**) die Member (**`int maxv`**) und Methoden (**`druckeboot()`**) der allgemeinsten Basisklasse **mindestens zweimal**. Das führt in der am meisten abgeleiteten Klasse zur Unmöglichkeit der späten Bindung von **`druckeboot()`** bzgl. einer Zeigervariablen **`boote *b=new segelbmitmotor(2);`** und zu nicht sinnvollen Mehrdeutigkeiten, die zum Teil mit Castings bzgl. der beiden übergeordneten Klassen aufgelöst werden können:

```
class boote {
          int maxv;
  public: boote(int i):maxv(i){cout<<"Konstruktor boote\n"; }

          virtual void druckeboot(){ cout<<"maxv= "<<maxv<<" Knoten\n"; }
};

class segelb : public boote {
          int segelf;
  public: segelb(int i):boote(111),segelf(i){cout<<"Konstruktor segelb\n";}

          virtual void druckeboot(){ cout<<"Segelfl.= "<<segelf<<" qm\n"; }
};

class motorb : public boote {
          int ps;
  public: motorb(int i):boote(222), ps(i){cout<<"Konstruktor motorb\n"; }

          virtual void druckeboot(){ cout<<"PS= "<<ps<<" PS\n"; }
};

class segelbmitmotor : public segelb, public motorb {
          int mannschaft;
  public: segelbmitmotor(int i):segelb(20),motorb(30), boote(15),
                                mannschaft(i){
               cout<<"Konstruktor segelbmitmotor\n"; }

          void druckeboot(){ cout<<"Bootsdaten:\n"
                             segelb::druckeboot();
                             motorb::druckeboot();
                             boote::druckeboot();
                             cout<<"Manschaft= "<<mannschaft<<" Mann\n";
          }
};
```

Einen Ausweg liefern die **virtuellen Basisklassen**. Diese sind dadurch charakterisiert, dass bei den Ableitungen aus der gemeinsamen Klasse **`boote`** der Zusatz **`virtual`** verwendet wird:
`class segelb : virtual public boote` und `class motorb : virtual public boote`

Dadurch bilden die Member (Datenmember und Methoden) der Basisklasse **`boote`** nur noch einen **einzigen Datesatz**, der für beide Ableitungslinien **identisch** ist.

In der Klasse **`segelbmitmotor`** gibt es dadurch nur noch **ein Member** **`maxv`** und **eine Methode** **`void druckeboot()`** .

Wichtig ist, dass bei der Konstruktordefinition der am **weitesten abgeleiteten Klasse** **`segelbmitmotor`** unbedingt die Basisklasse initialisiert wird:
`segelbmitmotor(int i)``:segelb(20),motorb(30),` **`boote(15)`**`, mannschaft(i){}`

Der Konstruktor der am weitesten abgeleitete Klasse **`segelbmitmotor`** wird im Falle virtueller Basisklassen zeitlich vor allen anderen Klassen initialisiert.

Initialisierungen von **`boote`** bei den Konstruktoren von **`segelb`** und **`motorb`** sind möglich und im Falle eines fehlenden Defaultkonstruktoren der Klasse **`boote`** sogar notwendig. Eine inhaltliche Initialisierung von **`boote`** erfolgt jedoch immer nur mit dem Konstruktor der am weitesten abgeleiteten Klasse:
`segelbmitmotor(int i)``:segelb(20),motorb(30),` **`boote(15)`**`, mannschaft(i){}` und das auch zeitlich vor allen anderen Konstruktoren. Andere Initialisierungen von Basis werden ignoriert. Fehlt die Initialisierung von **`boote`** beim Konstruktor von **`segelbmitmotor,`** dann wird an dieser Stelle der Defaultkonstruktor von **`boote`** gerufen.

Es ist nicht erlaubt, einen Zeiger auf eine virtuelle Basisklasse in einen Zeiger auf eine davon abgeleitete Klasse umzuwandeln.

```
#include <iostream>          // virt6.cpp
using namespace std;

class boote {
            int maxv;

    public: boote(int i):maxv(i){ cout<<"Konstruktor boote\n"; }

            virtual void druckeboot(){ cout<<"maxv= "<<maxv<<" Knoten\n"; }

            virtual ~boote(){};
};

class segelb : virtual public boote {
            int segelf;

    public: segelb(int i):boote(110),segelf(i){
                                  cout<<"Konstruktor segelb\n";
            }

            virtual void druckeboot(){ cout<<"Segelflaeche= "
                                       <<segelf
                                       <<" qm\n";
            }

            virtual ~segelb(){};
};

class motorb : virtual public boote {
            int ps;

    public: motorb(int i):boote(220), ps(i){cout<<"Konstruktor motorb\n";}

            virtual void druckeboot(){ cout<<"PS= "<<ps<<" PS\n"; }

            virtual ~motorb(){};
};

class segelbmitmotor : public segelb, public motorb {
            int mannschaft;

    public: segelbmitmotor(int i):segelb(20),motorb(30), boote(15),
                                  mannschaft(i){
                                  cout<<"Konstruktor segelbmitmotor\n"; }

            void druckeboot(){ cout<<"Bootsdaten:\n";
                               segelb::druckeboot();
                               motorb::druckeboot();
                               boote::druckeboot(); //eindeutig wg. virtual
                               cout<<"Manschaft= "<<mannschaft<<" Mannn";
            }

            virtual ~segelbmitmotor(){};
};

void main(){  segelbmitmotor sm(2);
              sm.druckeboot();                // fruehe Bindung

              boote *b=new segelbmitmotor(2);
              b->druckeboot();                // spaete Bindung
              delete b; b = 0;
}
```

Ausgabe:

```
Konstruktor boote
Konstruktor segelb
Konstruktor motorb
Konstruktor segelbmitmotor
Bootsdaten:
Segelflaeche= 20 qm
PS= 30 PS
maxv= 15 Knoten
Manschaft= 2 Mann
Konstruktor boote
Konstruktor segelb
Konstruktor motorb
Konstruktor segelbmitmotor
Bootsdaten:
Segelflaeche= 20 qm
PS= 30 PS
maxv= 15 Knoten
Manschaft= 2 Mann
```