Rein virtuelle Funktionen

Virtuelle Funktionen können dazu verwendet werden, auf Basisklassenebene verschiedene Spezialisierungen (in den abgeleiteten Klassen) einheitlich zu behandeln. Statt in der Basisklasse eine virtuelle Funktion zu definieren, kann auch eine *rein virtuelle* Funktion (pure virtual function) deklariert werden. Diese wird daran erkannt, daß für sie `=0` anstelle des Funktionsrumpfes steht.

Es können keine Objekte zu einer Klasse mit einer rein virtuellen Funktion erzeugt werden (aber es können Klassen von ihr abgeleitet werden).

Bsp.: Funktionen mit erster Ableitung, Voreinst. Differenzenquotient, selbstbeschreibend

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <limits>
#include <string>

using namespace std;

class Funktion {                    // abstrakte Klasse
  public:
    string text;
    string bemerkung;

    virtual double  f(double) = 0;   // rein virtuelle Funktion
    virtual double df(double x) {
      double h=max(1.0,abs(x))*sqrt(numeric_limits<double>::epsilon());
      return (f(x+h)-f(x))/h;
    }

    void auswert(double x) {
      cout << "Funktion f(x) = "  << text << ":" << endl
           <<  bemerkung          << endl
           << "x     = " << x     << endl
           << "f(x)  = " << f(x)  << endl
           << "f'(x) = " << df(x) << endl << endl;
    }
};

class Sin_mit_Abl : public Funktion {
  public:
    Sin_mit_Abl() { text="sin(x)"; bemerkung="Ableitung exakt"; }
    double  f(double x) { return sin(x); }
    double df(double x) { return cos(x); }
};

class Sin_ohne_Abl: public Funktion {
  public:
    Sin_ohne_Abl() { text="sin(x)"; bemerkung="Ableitung ueber Differenzenquot."; }
    double  f(double x) { return sin(x); }
};
```

```
int main()
{
  Sin_mit_Abl sin_mit_abl;
  sin_mit_abl.auswert(1.0);

  Sin_ohne_Abl sin_ohne_abl;
  sin_ohne_abl.auswert(1.0);

  return 0;
}
```

Ausgabe:

```
Funktion f(x) = sin(x):
Ableitung exakt
x     = 1
f(x)  = 0.841471
f'(x) = 0.540302

Funktion f(x) = sin(x):
Ableitung ueber Differenzenquotient
x     = 1
f(x)  = 0.841471
f'(x) = 0.540302
```

Abstrakte Klassen

Durch eine rein virtuelle Komponentenfunktion wird eine Klasse zu einer *abstrakten Klasse*, d.h. sie kann lediglich als Basisklasse anderer Klassen dienen und von ihr können keine Objekte gebildet werden. Sie kann auch nicht als Parametertyp oder Funktionsergebnistyp verwendet werden, jedoch sind Zeiger und Referenzen auf sie zulässig.

Bsp.:

```
#include <vector>
#include <list>
#include <iostream>
#include <sstream>
#include <string>
#include <cmath>

using namespace std;

class Figur {
  public:
    virtual double umfang() = 0;                      // rein virtuelle Funktion
    virtual string beschreibung() { return "Figur"; }
    virtual ~Figur() {};                              // Virtueller Destruktor
};

class Kreis : public Figur {
  private:
    double r;
  public:
    static double pi;
    Kreis(double r_=1) : r(r_) {}
    double umfang() { return 2*pi*r; }
    string beschreibung() {
      ostringstream os;
      os << "Kreis mit Radius " << r;
      return os.str();
    }
};

class Rechteck : public Figur {
  private:
    double a,b;
  public:
    Rechteck(double _a, double _b) : a(_a), b(_b) {}
    double umfang() { return 2*(a+b); }
    string beschreibung() {
      ostringstream os;
      os << "Rechteck mit Seitenlaengen " << a << " und " << b;
      return os.str();
    }
};
```

```
class KonvexesPolygon : public Figur {
  private:
    vector<double> x,y;
  public:
    KonvexesPolygon(const vector<double>& x_, const vector<double>& y_)
      : x(x_), y(y_) { }
    double umfang() {
      double u=0; int n=x.size();
      for (int i=0; i<n-1; ++i)
        u += sqrt((x[i+1]-x[i])*(x[i+1]-x[i])+(y[i+1]-y[i])*(y[i+1]-y[i]));
      u += sqrt((x[n-1]-x[0])*(x[n-1]-x[0])+(y[n-1]-y[0])*(y[n-1]-y[0]));
      return u;
    }
    string beschreibung() {
      ostringstream os;
      os << "Polygon aus " << x.size() << " Seiten";
      return os.str();
    }
};

double Kreis::pi = M_PI;

int main()
{
  list<Figur*> liste;
  Figur *k1p = new Kreis;
  Figur *k2p = new Kreis(2);
  liste.push_back(k1p);
  liste.push_back(k2p);
  liste.push_back(new Rechteck(1,2));

  int eckenzahl=131072;
  vector<double> x(eckenzahl), y(eckenzahl);
  for (int i=0; i<eckenzahl; ++i)
    { x[i]=cos(2*Kreis::pi*i/eckenzahl); y[i]=sin(2*Kreis::pi*i/eckenzahl);}
  liste.push_back(new KonvexesPolygon(x,y));

  for (list<Figur*>::iterator pos=liste.begin(); pos!=liste.end(); ++pos)
    cout << (**pos).beschreibung() << "  - Umfang: " << (**pos).umfang() << endl;

  // Speicherleck, falls kein virtueller Destruktor definiert
  for (int k=0; k<50; ++k) { Figur *f = new KonvexesPolygon(x,y); delete f; }
  return 0;
}
```

Ausgabe:

```
Kreis mit Radius 1  - Umfang: 6.28319
Kreis mit Radius 2  - Umfang: 12.5664
Rechteck mit Seitenlaengen 1 und 2  - Umfang: 6
Polygon aus 131072 Seiten  - Umfang: 6.28319
```