Funktionszeiger

Funktionszeiger verweisen auf Funktionen und kommen überall dort zum Einsatz, wo Funktionen nicht direkt verwendet werden dürfen, z.B. als Rückgabewert oder als Komponenten von Vektoren. Ursprünglich waren sie auch notwendig, wenn Funktionen als Parameter übergeben wurden. Funktionszeiger stehen zu Funktionen in einem ähnlichen Verhältnis wie Zeiger zu C-Vektoren.

Vereinbarungssyntax

Falls $T\ D$ einen Namen als "Datenstruktur aus T" vereinbart, dann vereinbart $T\ D(\ldots)$ denselben Namen als "Datenstruktur aus Funktion mit Ergebnistyp T".

Beispiele für Deklarationen:

`double sqr(double x)`	Funktion mit Ergebnistyp `double`
`char *gets(char *s)`	Funktion mit Ergebnistyp Zeiger auf `char`
`int (*fp)(double x)`	Zeiger auf Funktion mit Ergebnistyp `int`
`void (*signal(int sig, void (*func)(int)))(int)`	Funktion mit Ergebnistyp Zeiger auf Funktion ohne Ergebnis

Der Datentyp kann aus dem Deklarator durch Lesen von innen nach außen bestimmt werden.

Funktionsaufruf mit Funktionszeiger

Eine Funktion `f` kann auch durch einen Funktionszeiger, der auf sie verweist, aufgerufen werden.

Bsp:

```
double f(double x) { return x*x;}

int main()
{
           :
  double (*fp)(double);
  fp = &f;

  f(x)     ;   // korrekt
  (&f)(x)  ;   // korrekt
  fp(x)    ;   // korrekt
  (*fp)(x) ;   // korrekt
           :
}
```

Umwandlung von Funktionsnamen in Zeiger

Funktionsnamen werden in Funktionszeiger umgewandelt (Ausnahmen: `f()`, `&f`).

Vorsicht:
```
h(); /* Funktionsaufruf bei leerer Argumentliste */
h;   /* Funktionsadresse als Wert des Ausdrucks */
```

Diese Umwandlung findet insbesondere auch bei formalen und aktuellen Parametern vom Funktionstyp statt.

Bsp.: Sehnentrapezregel

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

double sehnentrapez(double a, double b, int n, double g(double))
{
  double s, h=(b-a)/n;
  int i;
  s = (g(a)+g(b))/2.0;
  for (i=1; i<n; ++i) s+=g(a+i*h); /* zulaessig waere auch s+=(*g)(a+i*h) */
  return s*h;
}

double f(double x)
{
  return exp(-x*x);
}

int main()
{
  double a,b;
  int i,j,n;

  typedef double (*Funcptr)(double);
  vector<Funcptr> funcptrvec = {f,sin,cos};

  typedef double (*Quadptr)(double,double,int,double(double));
  vector<Quadptr> quadptrvec = {sehnentrapez};

  cout << "a b n: "; cin >> a >> b >> n;

  for (i=0; i<funcptrvec.size(); ++i)
    cout << "Fkt. " <<  i << " mit Sehnentrapezregel: "
         << sehnentrapez(a,b,n,funcptrvec[i]) << endl;

  cout << endl;

  for (j=0; j<quadptrvec.size(); ++j)
    cout << "Fkt. f mit Quadraturformel " << j <<  ": "
         << quadptrvec[j](a,b,n,f) << endl;

  return 0;
}
```

- **Klasse zur Auswertung arithmetischer Ausdrücke einschl. Funktionsaufrufen**

```
/* Auswertung einfacher arithmetischer Ausdruecke
   Zwischenraum ist unzulaessig
   Fehlerbehandlung erfolgt nur rudimentaer */

       :
#include <sstream>
#include <cmath>
#include <map>

using namespace std;

typedef double (*Funcptr)(double);

class Ausdruck {

  private:
    string str;
    istringstream ein;
    static map<string,Funcptr> fmap;

    double WertFaktor()
    {
      double wert; char z;
      string funcname="";
      z = ein.peek();
      if ( isdigit(z) ) {
        ein >> wert;
      } else { // funcname(Ausdruck) - funcname kann fehlen
        while ( islower(z) ) { ein >> z; funcname+=z; z = ein.peek(); }
        if ( z=='(' ) {
          ein >> z;
          if (funcname=="") {
            wert = WertAusdruck();
          } else {
            if (fmap.find(funcname)!=fmap.end())
              wert = (*fmap[funcname])(WertAusdruck());  // Funktionsauswertung
            else
              fehler("unbekannte Funktion");
          }
          z = ein.peek();
          if ( z!=')' ) fehler("')' erwartet"); ein >> z;
        } else {
          fehler("unerwartetes Zeichen");
        }
      }
      return wert;
    }

    double WertAusdruck()
    { // Gegenueber Prog.I nur cin durch ein ersetzt }
```

```
    double WertTerm()
    { // Gegenueber Prog.I nur cin durch ein ersetzt }

    void fehler(string meldung)
    { // Verbesserte Fehlerbehandlung
      cout << "Fehler:" << endl << str << endl;
      int fehlerpos;
      if (ein.eof())
        fehlerpos = str.size();  // Fehler am Ende
      else
        fehlerpos = ein.tellg();
      for (int i=0; i<fehlerpos; ++i) cout << ' ';
      cout << '|' << " -- " << meldung << endl;
      exit(1);
    }

    void mapinit()
    { // Tabelle Funktionsnamen -> Funktionszeiger
      fmap={ {"sqrt",sqrt}, {"exp", exp }, {"log", log },
             {"sin", sin }, {"cos", cos }, {"tan", tan },
             {"asin",asin}, {"acos",acos}, {"atan",atan} };
    };

  public:
    Ausdruck(const Ausdruck& ausdruck) { // Kopierkonstruktor
      str=ausdruck.str; ein.str(str);    // Automatisch erzeugter nicht ausreichend,
    }                                    // weil Ein/Ausgabestroeme nicht kopierbar

    Ausdruck(string s) {                 // Konstruktor
      if (fmap.empty()) mapinit();       // Fkt.tabelle nur einmal initialisieren
      str=s; ein.str(s);
    }

  friend void auswerten(Ausdruck ausdruck) {
    double wert=ausdruck.WertAusdruck();
    if (ausdruck.ein.eof())
      cout << ausdruck.str << " = " << wert << endl;
    else
      ausdruck.fehler("unerwartete Zeichen am Ende");
  }
};

map<string,Funcptr> Ausdruck::fmap;      // Notwendig, weil fmap statische Komp.

int main()
{
  string s;
  cout << "Arithm. Ausdruck: ";
  getline(cin,s);
  Ausdruck ausdruck(s);
  auswerten(ausdruck);
  return 0;
}
```