HARALD NAHRSTEDT

C++ für Ingenieure

Einführung in die objektorientierte Programmierung

C++ Quellcodes zum Buch Kapitel 3

Erstell am	27.04.2009
Beschreibung	Diese Seiten enthalten den Quellcode aus dem Buch. Sie können kopiert und zur eigenen Programmierung weiter verwendet werden. Für die Ausführung wird jedoch keine Haftung übernommen.

Code 3-1 Deklaration der Schnittstelle der Klasse Stack

```
/* Stack.cc
   Dieses Modulelement
   deklariert die Schnittstellen (Interface)
   des Stacks
*/
#include "Stack.h"

void stack::push(int);
int stack::pop();
```

Code 3-2 Definition der Schnittstelle der Klasse Stack

```
/* Stack.h
   Dieses Modulelement (Header)
   definiert die Implementierungsschnittstelle
*/

#ifndef STACK_H
#define STACK_H

namespace stack
{
   const int maxSize = 200;
   int v[maxSize];
   int top = 0;

   void push(int s)
   {
      top = top+1;
      v[top] = s;
   }

   int pop()
   {
      int s;
      s = v[top];
      top = top-1;
      return s;
   }
}

#endif
```

Code 3-3 Anwendungsbeispiel der Klasse Stack

```
/* Stack.cpp
   Dieses Programm nutzt den Stack
   um 8 Ziffern aufsteigend zu speichern
   und auch wieder zu entfernen
*/
```

```
#include <iostream.h>
#include "Stack.cc"

int main()
{
   int i;

   for (i=1;i<9;i++) stack::push(i);
   for (i=1;i<9;i++)
      cout << "auf dem Stack: " << stack::pop() << endl;

   system("Pause");
}
```

Code 3-4 Deklaration der Klasse Scheibe

```
/* Scheibe.cc
   Dieses Modulelement
   deklariert die Schnittstellen (Interface)
   einer rotierenden Scheibe
*/

#include "Scheibe.h"
double scheibe::SigmaT (double, double, double, double);
double scheibe::SigmaR (double, double, double, double);
```

Code 3-5 Definition der Klasse Scheibe

```
/* Scheibe.h
   Dieses Modulelement (Header)
   definiert die Implementierungsschnittstelle
*/

#ifndef SCHEIBE_H
#define SCHEIBE_H

namespace scheibe
{
double p=0.00785;
double v=0.3;
double a1, a2, a3, a4, a5, a6;

double SigmaT (double w, double ra, double ri, double r)
{
   a1 = 3+v;
   a2 = ra*ra+ri*ri;
   a3 = ra*ra*ri*ri;
   a4 = 1+3*v;
   a5 = p/981*w*w/8;
   a6 = a5*(a1*(a2+a3/r/r)-a4*r*r);
   return a6;
}

double SigmaR (double w, double ra, double ri, double r)
```

```
{
   a1 = 3+v;
   a2 = ra*ra+ri*ri;
   a3 = ra*ra*ri*ri;
   a4 = p/981*w*w/8*a1;
   a6 = a4*(a2-a3/r/r-r*r);
   return a6;
}
}

#endif
```

Code 3-6 Anwendungsbeispiel der Klasse Scheibe

```
/* Scheibe.cpp
   Dieses Programm bestimmt
   die Spannungsverläufe
   in einer rotierenden Scheibe
   durch Fliehkräfte
*/

#include <iostream.h>
#include "Scheibe.cc"

int main()
{
   double w = 209;
   double r;
   double ra = 20;
   double ri = 0.5;
   FILE *puffer;
   puffer = fopen("C:\\Temp\\Scheibe.dat","w");

   for (r=0.5;r<20.1;r=r+0.1)
   {
      fprintf(puffer, "%2.1f \t%3.1f \t%3.1f \n", r,
      scheibe::SigmaT(w, ra, ri, r),
      scheibe::SigmaR(w, ra, ri, r));
   }
   system("Pause");
}
```

Code 3-7 Demoprogramm – Zugriff auf geschachtelte Struktur-Variable

```
/* Strukturen.cpp
   Das Programm zeigt die Nutzung
   einer geschachtelten Struktur
*/

#include <iostream.h>
#include <string>

struct GeoForm
{
```

```
   std::string Name;
   std::string Beschreibung;
   float Umfang;
   float Inhalt;
};

struct Kreis
{
   GeoForm KreisForm;
   float Radius;
   float Pi;
};

int main ()
{
    Kreis Kreis1;
    Kreis1.Pi=3.1415926;
    Kreis1.Radius=20;
    Kreis1.KreisForm.Name="Kreis";

    cout << Kreis1.KreisForm.Name << endl;
    cout << "Inhalt = " << Kreis1.Radius*Kreis1.Radius*Kreis1.Pi;
    cout << endl;

    system ("Pause");
}
```

Code 3-8 Anwendungsbeispiel Zufallszahlen – Flächenbestimmung nach der Monte-Carlo-Methode

```
/* MonteCarlo.cpp
   Flächenbestimmung
   nach der Monte-Carlo Methode
   mittels Pseudozufallszahlen
*/

#include <iostream.h>
#include <math.h>
#include <time.h>

int main()
{
   double x, y, z, AK, AB;
   int n, m, Max;

   cout << "Anzahl Punkte = ";
   cin >> Max;
   n=0;
   m=0;
   srand(time(NULL)); //Generator Initialisierung

   do
   {
   // RAND_MAX Maximalwert einer Pseudozufallszahl
      x = -80+rand()*160/RAND_MAX;
```

```
      y = -80+rand()*160/RAND_MAX;
      z = sqrt(x*x+y*y);
      if (z<=80)
      {
         n = n+1;
         if (y<=65) m=m+1;
      }
   }
   while (n<Max);
// Berechnung und Ausgabe
   AK = 80*80*M_PI;  //M_PI Standard-Kreiskonstante
   AB = AK*m/n;
   cout << "AK = " << AK << endl;
   cout << "AB = " << AB << endl;
   system("Pause");
}
```

Code 3-9 Anwendungsbeispiel Probabilistische Simulation – Warteschlange

```
/* Warteschlange.cpp
   Das Programm simuliert
   eine Maschinenwartung
*/

#include <iostream.h>
#include <math.h>
#include <time.h>
#include <string>

struct Maschine
{
   std::string Name;
   std::string Desc; // Beschreibung
   float aw;         // Ausfallwahrscheinlichkeit
   float rw;         // Reparaturwahrscheinlichkeit
   int ws;           // Warteschlange (0=nein/1=ja)
};

// Belegung der Strukturdaten
void Daten (Maschine Pro[])
{
   int i;

   for (i=0;i<10;i++) Pro[i].ws = 0;

   Pro[0].Name = "Maschine1";
   Pro[0].Desc = "Drehbank A";
   Pro[0].aw = 0.1;
   Pro[0].rw = 0.18;

   Pro[1].Name = "Maschine2";
   Pro[1].Desc = "Drehbank B";
   Pro[1].aw = 0.21;
   Pro[1].rw = 0.23;
```

```
   Pro[2].Name = "Maschine3";
   Pro[2].Desc = "Drehbank C";
   Pro[2].aw = 0.17;
   Pro[2].rw = 0.19;

   Pro[3].Name = "Maschine4";
   Pro[3].Desc = "Bohrmaschine A";
   Pro[3].aw = 0.14;
   Pro[3].rw = 0.14;

   Pro[4].Name = "Maschine5";
   Pro[4].Desc = "Bohrmaschine B";
   Pro[4].aw = 0.19;
   Pro[4].rw = 0.16;

   Pro[5].Name = "Maschine6";
   Pro[5].Desc = "Hobelbank A";
   Pro[5].aw = 0.18;
   Pro[5].rw = 0.21;

   Pro[6].Name = "Maschine7";
   Pro[6].Desc = "Hobelbank B";
   Pro[6].aw = 0.20;
   Pro[6].rw = 0.19;

   Pro[7].Name = "Maschine8";
   Pro[7].Desc = "Fraesmaschine A";
   Pro[7].aw = 0.2;
   Pro[7].rw = 0.16;

   Pro[8].Name = "Maschine9";
   Pro[8].Desc = "Fraesmaschine B";
   Pro[8].aw = 0.1;
   Pro[8].rw = 0.1;

   Pro[9].Name = "Maschine10";
   Pro[9].Desc = "Fraesmaschine C";
   Pro[9].aw = 0.11;
   Pro[9].rw = 0.09;
}

// Ausgabe der Strukturdaten
void Ausgabe (Maschine Pro[])
{
   int i;

   for (i=0;i<10;i++)
   {
      cout << i << "-te Maschine:" << endl;
      cout << "   Name: ";
      cout << Pro[i].Name << endl;
      cout << "   Beschreibung: ";
      cout << Pro[i].Desc << endl;
      cout << "   Ausfallwahr: ";
```

```
      cout << Pro[i].aw << endl;
      cout << "   Reparaturwahr: ";
      cout << Pro[i].rw << endl;
   }
   system("Pause");
}

int main ()
{
   int i;
   double t, dt, tMax, z;
   int iw;
   FILE *puffer;
   puffer = fopen("C:\\Temp\\Warteschlange.dat","w");

// Daten
   Maschine Pro[10];  // Produktionsmaschinen
   Daten (Pro);
   Ausgabe (Pro);

   dt = 0.1;
   tMax = 360;
   srand(time(NULL)); //Generator Initialisierung

// Simulation
   for (t=0;t<tMax;t=t+dt)
   {
      iw = 0;
      for (i=0;i<10;i++)
      {
      // Erzeugung einer Zufallszahl
      // im Intervall (0,1)
         z = double(rand()/double(RAND_MAX));
         if (Pro[i].ws==0)
         {
            if (z<=Pro[i].aw)
            {
               Pro[i].ws = 1;
               iw++;
            }
         }
         else
         {
            if (z<=Pro[i].rw) Pro[i].ws = 0;
            else iw++;
         }
      }
      cout << t << " / " << iw << endl;
      fprintf(puffer, "%2.2e \t%d \n", t, iw);
   }
   fclose(puffer);

   system("Pause");
}
```

Code 3-10 Demoprogramm – *Int* und *Double* teilen sich denselben Speicher

```
/* IntUndDouble.cpp
   Eine Int und eine Double Variable
   teilen sich den gleichen Speicher
*/

#include <iostream>
using namespace std;

struct IntUndDouble
{
   bool istInt;
   union
   {
      int Int;
      double Double;
   };
};

void setzeInt(IntUndDouble& s, int Value)
{
   s.Int = Value;
   s.istInt = false;
}

void setzeDouble(IntUndDouble& s, double Value)
{
   s.Double = Value;
   s.istInt = false;
}

void AusgabeValue(IntUndDouble const& s)
{
   if(s.istInt)
   {
      cout << "s.Int = " << s.Int << endl;
   }
   else
   {
      cout << "s.Double = " << s.Double << endl;
   }
}

int main()
{
   IntUndDouble s;
   setzeInt(s, 7);
   AusgabeValue(s);
   system("Pause");
}
```

Code 3-11 Demoprogramm – Zugriff auf die einzelnen Bytes eines int-Typs

```
/* UnionDemo.cpp
   Das Programm greift auf die
   einzelnen Bytes eines int-Typs zu
*/

#include <iostream>
using namespace std;

union Test
{
   int Int; //Annahme: sizeof(int) == 4
   struct
   {
      char byte1;
      char byte2;
      char byte3;
      char byte4;
   }  Bytes;
};

int main()
{
   Test t;
   t.Int = 888888888;

   cout << "Der int-Wert " << t.Int
        << " besteht aus folgenden Bytes:\n";
   cout << "Byte 1: " << t.Bytes.byte1 << endl;
   cout << "Byte 2: " << t.Bytes.byte2 << endl;
   cout << "Byte 3: " << t.Bytes.byte3 << endl;
   cout << "Byte 4: " << t.Bytes.byte4 << endl;

   system("Pause");
}
```

Code 3-12 Anwendungsbeispiel - Permutationen natürlicher Zahlen

```
/* Permutationen.cpp
Das Programm erzeugt
Permutationen natürlicher Zahlen
*/

#include <iostream>
using namespace std;

void Permutation (int k, int n, int x[]);
void Ausgabe (int n, int x[]);
int main()
{
int i, j, n;
cout << "Anzahl = ";
cin >> n;

int x[n];
```

```
for (i=0;i<n;i++)
x[i] = i;
Permutation(0, n, x);
system("Pause");
}

void Permutation (int k, int n, int x[])
{
int i, y;
y = x[k];
for (i=k;i<n;i++)
   {
x[k] = x[i];
x[i] = y;
if (k<n-1)
Permutation (k+1, n, x);
else
Ausgabe (n, x);
x[i] = x[k];
   }
x[k] = y;
}

void Ausgabe (int n, int x[])
{
int i;
for (i=0;i<n;i++)
cout << x[i];
cout << endl;
}
```

Code 3-13 Anwendungsbeispiel Engpassproblem—Fließbandarbeit

```
/* Engpass.cpp
   Das Programm loesst das Engpassproblem
   für 6 Stationen mit jeweils 6 Personen
*/

#include <iostream>
using namespace std;

// Definition globaler Variabler
int n = 6;
int a[6][6];
int Max;

//Deklaration der Funktionen
void Daten ();
void Permutation (int k, int x[]);
void Ausgabe (int x[]);

// Hauptprogramm
int main()
```

```
{
   int i;

// Daten lesen
   Daten ();

   int x[6];
   for (i=0;i<n;i++)
      x[i] = i;

   Permutation(0, x);
   system("Pause");
}

// Testdaten
void Daten ()
{
   a[0][0] = 7;
   a[0][1] = 5;
   a[0][2] = 11;
   a[0][3] = 13;
   a[0][4] = 14;
   a[0][5] = 9;
   a[1][0] = 3;
   a[1][1] = 6;
   a[1][2] = 15;
   a[1][3] = 6;
   a[1][4] = 3;
   a[1][5] = 11;
   a[2][0] = 10;
   a[2][1] = 6;
   a[2][2] = 3;
   a[2][3] = 1;
   a[2][4] = 2;
   a[2][5] = 7;
   a[3][0] = 4;
   a[3][1] = 7;
   a[3][2] = 5;
   a[3][3] = 1;
   a[3][4] = 6;
   a[3][5] = 8;
   a[4][0] = 5;
   a[4][1] = 7;
   a[4][2] = 8;
   a[4][3] = 3;
   a[4][4] = 10;
   a[4][5] = 6;

   a[5][0] = 4;
   a[5][1] = 5;
   a[5][2] = 3;
   a[5][3] = 6;
   a[5][4] = 12;
   a[5][5] = 9;
}
```

```
// Erstellung der Permutation
void Permutation (int k, int x[])
{
   int i, y;
   y = x[k];
   for (i=k;i<n;i++)
   {
      x[k] = x[i];
      x[i] = y;
      if (k<n-1)
         Permutation (k+1, x);
      else
         Ausgabe (x);
      x[i] = x[k];
   }
   x[k] = y;
}

void Ausgabe (int x[])
{
   int i, z=0;

   for (i=0;i<n;i++)
      z = z+a[i][x[i]];

   if (z>Max)
   {
      Max = z;
      for (i=0;i<n;i++)
         cout << x[i];
      cout << " = " << z << endl;
   }
}
```

Code 3-14 Demoprogramm – Kopie der Struktur als Parameter

```
/* Dreieck1.cpp
   Übergabe der Struktur als Kopie
*/

#include <iostream>
using namespace std;

struct Dreieck {int a;int b;int c;};

int Umfang(struct Dreieck x)
{
   return x.a+x.b+x.c;
};

int main ()
{
   struct Dreieck D;
```

```
   D.a = 3;
   D.b = 4;
   D.c = 5;

   cout << "Umfang = ";
   cout << Umfang(D) << endl;
   system ("Pause");
}
```

Code 3-15 Demoprogramm – Zeiger auf die Struktur als Parameter

```
/* Dreieck2.cpp
   Übergabe der Struktur als Zeiger
*/

#include <iostream>
using namespace std;

struct Dreieck {int a;int b;int c;};

int Umfang(struct Dreieck *x)
{
   return (*x).a+(*x).b+(*x).c;
};

int main ()
{
   struct Dreieck D, *pD;

   pD = &D;
   D.a = 3;
   D.b = 4;
   D.c = 5;

   cout << "Umfang = ";
   cout << Umfang(pD) << endl;
   system ("Pause");
}
```

Code 3-16 Anwendungsbeispiel – PERT-Netzplan

```
/* Pert
   Das Programm ermittelt Projektzeiten und Engpässe
   nach der Methode PERT der Netzplantechnik
*/

#include <iostream.h>
#include <string.h>
#include <math.h>

struct Knoten
{
   char Taetigkeit[20];
```

```
   int von;
   int bis;
   float OZeit;
   float WZeit;
   float PZeit;
   char Ereignis[20];
   float Dauer;
   float Varianz;
   float SpaetestensStartenUm;
   float SpaetestensEndenUm;
   float FruehesterStartUm;
   float SpaetesterStartUm;
   float FruehestesEndeUm;
   float SpaetestesEndeUm;
   float Schlupf;
};

Knoten Knoten[9];

void Daten (void);
void Ausgabe (void);

int main()
{
   int i;
   double S[9], F[9];
   double L, V, SL;

// Daten einlesen
   Daten ();

// Erwartete Dauer und Varianz
   for (i=0;i<9;i++)
   {
      Knoten[i].Dauer =
         (Knoten[i].OZeit+4*Knoten[i].WZeit+Knoten[i].PZeit)/6;
      Knoten[i].Varianz =
         (Knoten[i].PZeit-Knoten[i].OZeit)/6;
   S[i] = 0;
   F[i] = 0;
   }

// Auffinden der frühesten Startzeiten
   for (i=0;i<9;i++)
   {
      if (S[Knoten[i].bis]<S[Knoten[i].von]+Knoten[i].Dauer)
         S[Knoten[i].bis] = S[Knoten[i].von]+Knoten[i].Dauer;
   }
   F[Knoten[8].bis] = S[Knoten[8].bis];

// Auffinden der spätesten Endzeiten
   for (i=9;i>0;--i)
   {
      if (F[Knoten[i].von]==0 ||
          F[Knoten[i].von]>F[Knoten[i].bis]-Knoten[i].Dauer)
```

```
            F[Knoten[i].von] = F[Knoten[i].bis]-Knoten[i].Dauer;
   }

// Suchen des kritischen Pfades
   L=0;
   V=0;
   for (i=9;i>0;--i)
   {
      Knoten[i].Schlupf =
         F[Knoten[i].bis]-S[Knoten[i].von]-Knoten[i].Dauer;
      if (Knoten[i].Schlupf<=0.1)
      {
         strcpy(Knoten[i].Ereignis,"Kritisch!");
         Knoten[i].SpaetestensStartenUm = S[Knoten[i].von];
         Knoten[i].SpaetestensEndenUm = F[Knoten[i].bis];
      // Aufsummieren von Pfadlänge und Varianz
         if (L<F[Knoten[i].bis])
            L = F[Knoten[i].bis];
         V = V+Knoten[i].Varianz*Knoten[i].Varianz;
      }
      else
      {
         Knoten[i].FruehesterStartUm = S[Knoten[i].von];
         Knoten[i].SpaetesterStartUm =
            F[Knoten[i].bis]-Knoten[i].Dauer;
         Knoten[i].FruehestesEndeUm =
            S[Knoten[i].von]+Knoten[i].Dauer;
         Knoten[i].SpaetestesEndeUm = F[Knoten[i].bis];
      }
   }

// Ausgabe
   cout << "Kritischer Pfad = " << L << endl;
   cout << "Pfadabweichung  = " << sqrt(V) << endl;
   system("Pause");
   Ausgabe();
   system("Pause");
}

void Daten()
{
   int i;
// Die Tätigkeitsbeschreibungen werden
// hier vereinfacht durch einfache
// Beziehungsangaben der Knoten
   strcpy(Knoten[0].Taetigkeit,"1-2");
   strcpy(Knoten[1].Taetigkeit,"1-3");
   strcpy(Knoten[2].Taetigkeit,"1-4");
   strcpy(Knoten[3].Taetigkeit,"2-5");
   strcpy(Knoten[4].Taetigkeit,"3-6");
   strcpy(Knoten[5].Taetigkeit,"4-6");
   strcpy(Knoten[6].Taetigkeit,"5-7");
   strcpy(Knoten[7].Taetigkeit,"6-7");
   strcpy(Knoten[8].Taetigkeit,"7-8");
```

```
   Knoten[0].von = 1;
   Knoten[1].von = 1;
   Knoten[2].von = 1;
   Knoten[3].von = 2;
   Knoten[4].von = 3;
   Knoten[5].von = 4;
   Knoten[6].von = 5;
   Knoten[7].von = 6;
   Knoten[8].von = 7;

   Knoten[0].bis = 2;
   Knoten[1].bis = 3;
   Knoten[2].bis = 4;
   Knoten[3].bis = 5;
   Knoten[4].bis = 6;
   Knoten[5].bis = 6;
   Knoten[6].bis = 7;
   Knoten[7].bis = 7;
   Knoten[8].bis = 8;

   Knoten[0].OZeit = 2;
   Knoten[1].OZeit = 3;
   Knoten[2].OZeit = 4;
   Knoten[3].OZeit = 2;
   Knoten[4].OZeit = 0;
   Knoten[5].OZeit = 3;
   Knoten[6].OZeit = 1.5;
   Knoten[7].OZeit = 1;
   Knoten[8].OZeit = 2;

   Knoten[0].WZeit = 3;
   Knoten[1].WZeit = 4;
   Knoten[2].WZeit = 6;
   Knoten[3].WZeit = 4;
   Knoten[4].WZeit = 0;
   Knoten[5].WZeit = 5;
   Knoten[6].WZeit = 4;
   Knoten[7].WZeit = 3;
   Knoten[8].WZeit = 5;

   Knoten[0].PZeit = 5;
   Knoten[1].PZeit = 5;
   Knoten[2].PZeit = 9;
   Knoten[3].PZeit = 7;
   Knoten[4].PZeit = 0;
   Knoten[5].PZeit = 8.5;
   Knoten[6].PZeit = 7;
   Knoten[7].PZeit = 7;
   Knoten[8].PZeit = 9;

   for (i=0;i<9;i++)
      strcpy(Knoten[i].Ereignis, "n. kritisch!");
}

void Ausgabe ()
```

```
{
   int i;
   printf("\nPfad \tEreignis \tDauer \tVarianz \n");
   for (i=0;i<9;i++)
   {
      printf("%s \t%s \t%3.3f \t%3.3f \n", Knoten[i].Taetigkeit,
         Knoten[i].Ereignis, Knoten[i].Dauer, Knoten[i].Varianz);
   }
}
```

Code 3-17 Demoprogramm – Verkettete Liste

```
/* VerketteteListe.cpp
   Zunächst wird eine
   verkettete Liste erstellt
   und diese dann umgekehrt
   wieder ausgegeben
*/

#include <iostream>
using namespace std;

struct ListenKnoten
{
   int data;             // Daten
   ListenKnoten *next;   // Zeiger auf den Nachfolger
};

ListenKnoten *Anker = 0; // Anfang der Liste

int main()
{
   int Inhalt;
   ListenKnoten *node, *old;

   // Erstelle Liste mit Werten
   do
   {
      cout << "Zahl eingeben (0=Ende)" << endl;
      cin >> Inhalt;
      if (Inhalt)
      {
         // Neues Element für die Liste erzeugen:
         ListenKnoten *node = new ListenKnoten;
         node->data = Inhalt; // Besetze die Daten
         node->next = Anker;  // Hänge die bisherige Liste an
         Anker = node;        // Setze den Anfangspunkt hierher
      }
   } while (Inhalt);

   // Ausgabe der Liste in umgekehrter Reihenfolge
   // und Löschen der ausgegebenen Elemente
   while (Anker)    // ungleich Null! Die Liste ist nicht leer!
   {
```

```
      cout << Anker->data << endl;
      old = Anker;         // Sicherung zum späteren Löschen
      Anker = Anker->next; // nächstes Element nach vorziehen
      delete old;          // Löschung des ausgegebenen Elements
   }

   system("Pause");
}
```

Code 3-18 Anwendungsbeispiel Ameisenalgorithmus – Maschinenbelegung

```
/* Ameisen.cpp
   Bestimmung einer optimalen Maschinenbelegung
   mit Hilfe des Ameisen-Algorithmus
*/

#include <iostream>
#include <iomanip>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

using namespace std;
int p=5, m=3;   // Anzahl Produkte; Maschinen
int M[5][3];    // Maschinenstunden-Matrix

// Beispieldaten
void Daten ()
{
   M[0][0] = 5;
   M[0][1] = 3;
   M[0][2] = 8;
   M[1][0] = 5;
   M[1][1] = 3;
   M[1][2] = 4;
   M[2][0] = 12;
   M[2][1] = 4;
   M[2][2] = 2;
   M[3][0] = 4;
   M[3][1] = 2;
   M[3][2] = 7;
   M[4][0] = 8;
   M[4][1] = 2;
   M[4][2] = 9;
}

int main()
{
   int i, j, k, l, r, v;
   int a, b;
   int g=100;
   int n=500;
   int z[n], y[p][2], f[p], u[m], s[n];
```

```
int q, su, Min;
float w, x;
int Folge, PM[1000][3], pi=0;

srand(time(NULL)); //Zufallsgenerator Initialisierung

Daten ();

for (i=0;i<g;i++)
{
// für jede Pseudo-Ameise der Gruppe
// suche Produktfolge
   for (j=0;j<n;j++)
   {
   // Startbelegung
   // ein Produkt kann nur einmal gesetzt werden
      for (k=0;k<p;k++)
      {
         y[k][0]=k;
         y[k][1]=1; // Häufigkeit
      }

   // Bestimmung der Produktfolge
      for (k=0;k<p;k++)
      {
         q=0;
         do
         {
         // Berücksichtigung der Häufigkeiten
         // in der Pheromon-Matrix
            if (k>0 && pi>0)
            {
               for (l=0;l<p;l++)
               {
                  if (y[l][0]>0)
                  {
                  // bisherige Folge
                     Folge=0;
                     for (r=0;r<l;r++)
                     {
                        if (r==0)
                        {
                           Folge=f[r]+1;
                        }
                        else
                        {
                           Folge=Folge*10+f[r]+1;
                        }
                     }
                     y[l][1] = 1;
                     for (r=0;r<pi;r++)
                     {
                     // Suche gleiche
                     // Produktfolgen (Startfolgen)
                     // in der Pheromon-Matrix
```

```
                        a = PM[r][0];
                        b = a/pow(10, p-l);
                        if (Folge==b)
                        {
                           y[l][1] = y[l][1]+PM[r][2];
                        }
                     }
                  }
               }
            }

         // --- Wahrscheinlichkeitsverteilung ---

         // Bildung der Summe aller Häufigkeiten
            su=0;
            for (l=0;l<p;l++)
            {
               su=su+y[l][1];
            }

         // Erzeugung einer Zufallszahl im Intervall (0,1)
         // und deren Scalierung auf die Summe der Häufigkeiten
            x = double(rand()/double(RAND_MAX));
            w = x*su;

         // Bestimmung des Produkts aus den Häufigkeiten
         // Gleichverteilung der Pseudozufallszahlen
         // vorausgesetzt
            su=0;
            v=-1;
            for (l=0;l<p;l++)
            {
               su=su+y[l][1];
               if (su>=w&&v==-1)
               {
                  v = l;
               }
            }

         // Produkt noch nicht gewählt
            if (y[v][0]>-1)
            {
               y[v][0] = -1;
               y[v][1] = 0;
               f[k] = v;
               q = 1;
            }
         }
         while (q=0);
      }

   // Produktfolge in Zahl umwandeln
      Folge=0;
      for (k=0;k<p;k++)
      {
```

```
            if (k==0)
            {
               Folge=f[k]+1;
            }
            else
            {
               Folge=Folge*10+f[k]+1;
            }
         }
      // Produktfolge im Vektor merken
         z[j] = Folge;

      // Bestimmung der Durchlaufzeiten
         for (l=0;l<m;l++)
            u[l]=0;
         for (k=0;k<p;k++)
         {
            for (l=0;l<m;l++)
            {
               if (k==0)
               {
            // beim ersten Produkt
            // starten die Maschinenzeiten
                  u[l]=M[f[k]][l];
               }
               else
            // bei jedem weiteren Produkt
            // addieren sich die Produktionszeiten
               {
                  if (l>0)
                  {
                     if (u[l-1]<u[l])
                     {
                     // Belegungszeiten ohne Leerzeiten
                        u[l] = u[l]+M[f[k]][l];
                     }
                     else
                     {
                     // Belegungszeiten mit Leerzeiten
                        u[l] = u[l-1]+M[f[k]][l];
                     }
                  }
                  else
                  {
                     u[l]=u[l]+M[f[k]][l];
                  }
               }
            }
         }
      // Produktszeiten im Vektor merken
         s[j]=u[m-1];
      }

   // Beurteilung der Gruppe
   // Nur die minimalsten Werte werden eingetragen
```

```
   // Zuerst Bestimmung des Minimalwerts
      for (j=0;j<n;j++)
      {
         if (j==1)
            Min=s[j];
         else
            if (s[j]<Min&&s[j]!=0)
               Min=s[j];
      }

   // Eintrag der Min-Werte in die Pheromon-Matrix PM
      for (j=0;j<n;j++)
      {
         if (s[j]==Min)
         {
            q=0;
            for (k=0;k<pi;k++)
            {
               if (PM[k][0]==z[j])
               {
                  PM[k][2]++;
                  q = 1;
               }
            }
            if (q==0)
            {
               PM[pi][0] = z[j]; // Ziffernfolge
               PM[pi][1] = s[j]; // Beurteilung
               PM[pi][2] = 1;    // Pheromonanteil
               pi++;
            }
         }
      }
   }

// Ausgabe der Pheromon-Matrix
   for (i=0;i<pi;i++)
   {
      cout << PM[i][0] << "\t";
      cout << PM[i][1] << "\t";
      cout << PM[i][2] << endl;
   }
   system("Pause");
}
```

Code 3-19 Demoprogramm – anfordern und freigeben von Feld-Speicher

```
/* Zufall.cpp
   Das Programm fordert Speicher für 100 int-Typen
   erzeugt 100 Zufallszahlen
   gibt den Speicher wieder frei
   und gibt die noch vorhandenen Zahlen aus
*/
#include <iostream>
```

```
using namespace std;

int main()
{
    int i;
    int* Zufall = new int[100]; // Array mit 100 Elementen erzeugen

    for (i=0; i<100; i++)       // Array durchlaufen
    {
        Zufall[i] = rand();     // Zufallszahlen erzeugen
    }
    delete[] Zufall;            // Freigabe des Speichers

    for (i=0; i<100; i++)       // Ausgabe
    {
        cout << Zufall[i] << endl;
    }

    system("Pause");
}
```

Code 3-20 Anwendungsbeispiel Evolutionsstrategie – Optimierung eines Stabwerks

```
/* Evolution.cpp
   Stabwerksoptimierung
   nach der Evolutionsstrategie
*/
#include <iostream>
#include <math.h>
using namespace std;

void Daten ();

int main()
{
   double Ar [100][8]; // Array mit 100x12 Elementen erzeugen
   double x1a, y1a, x1n, y1n, dx1, dy1;
   double x2a, y2a, x2n, y2n, dx2, dy2;
   int i, j, m1, m2, m3;
   double a, x;
   double l1a, l2a, l3a, l4a, l5a;
   double l1n, l2n, l3n, l4n, l5n;
   double g12a, g12n, g34a, g34n;

   srand(time(NULL)); //Zufallsgenerator Initialisierung

   for (i=0; i<50; i++)       // Auswertungsschleife
   {
      x1a = 0;
      y1a = 0;
      x2a = 0;
      y2a = 0;
      a = 100;
      m1 = 0;
```

```
      m2 = 0;
      m3 = 0;
      l1a = a;
      l2a = a;
      l3a = a;
      l4a = a;
      l5a = int(sqrt(2)*a);

      do
      {
      // 1. Knoten
         g12a = l1a+l2a;
         x = double(rand()/double(RAND_MAX));
         dx1 = 3*x-1.5;
         x = double(rand()/double(RAND_MAX));
         dy1 = 3*x-1.5;
         x1n = x1a+dx1;
         y1n = y1a+dy1;
         l1n = int(sqrt((a-y1n)*(a-y1n)+x1n*x1n));
         l2n = int(sqrt((a+x1n)*(a+x1n)+y1n*y1n));
         g12n = l1n+l2n;
         m1++;
         if (g12n<g12a) m1=0;
      // 2. Knoten
         g34a = l3a+l4a;
         x = double(rand()/double(RAND_MAX));
         dx2 = 3*x-1;
         x = double(rand()/double(RAND_MAX));
         dy2 = 3*x-1;
         x2n = x2a+dx2;
         y2n = y2a+dy2;
         l3n = int(sqrt((a-x1n+x2n)*(a-x1n+x2n)+
               (y1n-y2n)*(y1n-y2n)));
         l4n = int(sqrt((a-y2n)*(a-y2n)+x2n*x2n));
         g34n = l3n+l4n;
         m2++;
         if (g34n<g34a) m2=0;
      // letzte Verbindung
         l5n = int(sqrt((a-x2n)*(a-x2n)+(a-y2n)*(a-y2n)));
         m3++;
         if (m1==0 && m2==0)
         {
            x1a = x1n;
            y1a = y1n;
            l1a = l1n;
            l2a = l2n;
            g12a = g12n;
            x2a = x2n;
            y2a = y2n;
            l3a = l3n;
            l4a = l4n;
            g34a = g34n;
            l5a = l5n;
            m3 = 0;
         }
```

```
      }
      while (m3<10000);
      Ar [i][0] = l1a;
      Ar [i][1] = l2a;
      Ar [i][2] = l3a;
      Ar [i][3] = l4a;
      Ar [i][4] = g12a;
      Ar [i][5] = g34a;
      Ar [i][6] = l5a;
      Ar [i][7] = g12a+g34a+l5a;
   }

// Ausgabe der Matrix
   for (i=0;i<50;i++)
   {
      for (j=0;j<8;j++)
         cout << Ar[i][j] << "\t";
      cout << endl;
   }
    system("Pause");
}
```