```
// **********



// Jacobi.C

// Berechnet alle Eigewerte und Eigenvektoren einer reellen symmetrischen
// Matrix.



// **********



#include <math.h>
#include <stdio.h>
#include <stdlib.h>



// **********



// m -- m -- m -- m -- ... -- m

const int N = 10;

// Symmetrische Matrix deren Eigenwerte und Eigenvektoren berechnet werden;
// wird überschrieben; am Ende stehen die Eigenwerte auf der Diagonalen.
double A[N][N];

// "Eigenvektormatrix" (Produkt von Jacobi-Rotationen), deren Spalten am Ende
// die Eigenvektoren von A sind.
double V[N][N];

const double epsilon = 0.0000000001 * 0.0000000001;



// **********



int main(int argc, char **argv)
{
  FILE *file1;
  int i1, i2, i3;
  char string1[10000];


  // *****


  // Initialisiere die Massen/Kraft-Matrix.

  for(i1 = 0; i1 < N; i1++)
    {
      for(i2 = 0; i2 < N; i2++)
        A[i1][i2] = 0.0;
    }

  for(i1 = 0; i1 < N-1; i1++)
    {
      A[i1  ][i1  ] += 1.0;
      A[i1  ][i1+1] -= 1.0;
      A[i1+1][i1  ] -= 1.0;
      A[i1+1][i1+1] += 1.0;
    }

  // /*
  for(i1 = 0; i1 < N; i1++)
    {
      for(i2 = 0; i2 < N; i2++)
        fprintf(stderr, "%+4.2lf   ", A[i1][i2]);

      fprintf(stderr, "\n");
    }
  // */


  // Initialisiere die "Eigenvektormatrix".

  for(i1 = 0; i1 < N; i1++)
    {
      for(i2 = 0; i2 < N; i2++)
        {
          if(i1 == i2)
            V[i1][i2] = 1.0;
          else
            V[i1][i2] = 0.0;
        }
    }


  // *****


  // Jacobi-Methode (Laufzeitverhalten kann verbessert werden, wenn nur eines
  // der beiden Dreiecke modifiziert wird [Symmetrie ausnutzen]).

  int ctr = 0;

  while(1)
    {
      // Summe der Quadrate der off-diagonal Elemente.

      double S = 0.0;

      for(i1 = 0; i1 < N; i1++)
        {
          for(i2 = 0; i2 < i1; i2++)
            S += pow(A[i1][i2], 2.0);
        }

      S *= 2.0;

      fprintf(stderr, "  S = %.5e.\n", S);

      if(S <= epsilon)
        break;

      // *****

      ctr++;

      fprintf(stderr, "sweep %4d ...\n", ctr);

      // Nächster Sweep ...

      for(i1 = 0; i1 < N; i1++)
        {
          for(i2 = 0; i2 < i1; i2++)
            {
              // !!!!!!!!!!
              // !!!!!!!!!!
              // !!!!!!!!!!
              // Hier sollte noch getestet werden, ob A[i1][i2] sehr klein,
              // ob theta sehr groß, etc. ist.
              // !!!!!!!!!!
              // !!!!!!!!!!
```

```
143                 // !!!!!!!!!!
144
145                 // theta
146
147                 double theta = 0.5 * (A[i2][i2] - A[i1][i1]) / A[i1][i2];
148
149                 // t
150
151                 double t = 1.0 / (fabs(theta) + sqrt(pow(theta, 2.0) + 1.0));
152
153                 if(theta < 0.0)
154                   t = -t;
155
156                 // c, s
157
158                 double c = 1.0 / sqrt(pow(t, 2.0) + 1.0);
159                 double s = t * c;
160
161                 // tau
162
163                 double tau = s / (1.0 + c);
164
165                 // Jacobi-Rotation
166
167                 // Matrix A.
168
169                 double A_pp = A[i1][i1] - t * A[i1][i2];
170                 double A_qq = A[i2][i2] + t * A[i1][i2];
171
172                 double A_rp[N], A_rq[N];
173
174                 for(i3 = 0; i3 < N; i3++)
175                   {
176                     if(i3 != i1 && i3 != i2)
177                       {
178                         A_rp[i3] = A[i3][i1] - s * (A[i3][i2] + tau * A[i3][i1]);
179                         A_rq[i3] = A[i3][i2] + s * (A[i3][i1] - tau * A[i3][i2]);
180                       }
181                   }
182
183                 A[i1][i2] = 0.0;
184                 A[i2][i1] = 0.0;
185
186                 A[i1][i1] = A_pp;
187                 A[i2][i2] = A_qq;
188
189                 for(i3 = 0; i3 < N; i3++)
190                   {
191                     if(i3 != i1 && i3 != i2)
192                       {
193                         A[i3][i1] = A_rp[i3];
194                         A[i1][i3] = A_rp[i3];
195
196                         A[i3][i2] = A_rq[i3];
197                         A[i2][i3] = A_rq[i3];
198                       }
199                   }
200
201                 // "Eigenvektormatrix" V.
202
203                 double V_rp[N], V_rq[N];
204
205                 for(i3 = 0; i3 < N; i3++)
206                   {
207                     V_rp[i3] = V[i3][i1] - s * (V[i3][i2] + tau * V[i3][i1]);
208                     V_rq[i3] = V[i3][i2] + s * (V[i3][i1] - tau * V[i3][i2]);
209                   }
210
211                 for(i3 = 0; i3 < N; i3++)
212                   {
213                     V[i3][i1] = V_rp[i3];
214                     V[i3][i2] = V_rq[i3];
215                   }
216               }
217           }
218
219         // /*
220         for(i1 = 0; i1 < N; i1++)
221           {
222             for(i2 = 0; i2 < N; i2++)
223               fprintf(stderr, "%+4.2lf   ", A[i1][i2]);
224
225             fprintf(stderr, "\n");
226           }
227         // */
228       }
229
230
231     // *****
232
233
234     for(i1 = 0; i1 < N; i1++)
235       {
236         fprintf(stderr, "\n\\lambda_%02d = %+10.6lf.\n", i1, A[i1][i1]);
237
238         fprintf(stderr, "v_%02d = ( ");
239
240         for(i2 = 0; i2 < N; i2++)
241           {
242             fprintf(stderr, "%+5.2lf", V[i2][i1]);
243
244             if(i2 < N-1)
245               fprintf(stderr, " , ");
246             else
247               fprintf(stderr, " ).\n");
248           }
249
250         // Teste Normierung:
251
252         double norm = 0.0;
253
254         for(i2 = 0; i2 < N; i2++)
255           norm += pow(V[i2][i1], 2.0);
256
257         if(fabs(norm - 1.0) > 0.0000000001)
258           {
259             fprintf(stderr, "Error: Eigenvektor nicht korrekt normiert.\n");
260             exit(EXIT_FAILURE);
261           }
262       }
263
264
265     // *****
266
267
268     return EXIT_SUCCESS;
269 }
270
271
272
273 // **********
```