```c
// RK.C

// Löse DGl-System
//
//   \vec{\dot{y}}(t) = \vec{f}(\vec{y}(t),t)
//
// mit Anfangsbedingungen
//
//   \vec{y}(t=0) = \vec{y}_0 .
//
// Hier speziell für den harmonischen Oszillator mit Potential
//
//   V(x) = m \omega^2 x^2 / 2 .

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

#define __EULER__
// #define __RK_2ND__
// #define __RK_3RD__
// #define __RK_4TH__

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

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

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

const int N = 2;  // Anzahl der Komponenten von \vec{y} bzw. \vec{f}.

const double omega = 1.0;  // Frequenz.

const int num_steps = 10000;  // Anzahl der RK-Schritte.
const double tau = 0.1;  // Schrittweite.

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

double y[N][num_steps+1];  // Diskretisierte "Bahnkurven".

double y_0[N] = { 1.0 , 0.0 };  // Anfangsbedingungen.

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

int main(int argc, char **argv)
{
  int i1, i2;

  // *****

  // Initialisiere "Bahnkurven" mit Anfangsbedingungen.

  for(i1 = 0; i1 < N; i1++)
    y[i1][0] = y_0[i1];

  // *****

  // Führe Euler/RK-Schritte aus.

  for(i1 = 1; i1 <= num_steps; i1++)
    {
      // 1D-HO:
      // y(t) = (x(t) , \dot{x}(t)) ,
      // \dot{y}(t) = f(y(t),t) = (\dot{x}(t) , F/m)
      //   wobei Kraft F = -m \omega^2 x(t) .

#ifdef __EULER__

      // Berechne k1 = f(y(t),t) * tau.

      double k1[N];

      k1[0] = y[1][i1-1]   * tau;
      k1[1] = -pow(omega, 2.0) * y[0][i1-1]   * tau;

      // *****

      for(i2 = 0; i2 < N; i2++)
        y[i2][i1] = y[i2][i1-1] + k1[i2];

#endif

#ifdef __RK_2ND__

      // Berechne k1 = f(y(t),t) * tau.

      double k1[N];

      k1[0] = y[1][i1-1]   * tau;
      k1[1] = -pow(omega, 2.0) * y[0][i1-1]   * tau;

      // *****

      // Berechne k2 = f(y(t)+(1/2)*k1 , t+(1/2)*tau) * tau.

      double k2[N];

      k2[0] = (y[1][i1-1] + 0.5*k1[1])   * tau;
      k2[1] = -pow(omega, 2.0) * (y[0][i1-1] + 0.5*k1[0])   * tau;

      // *****

      for(i2 = 0; i2 < N; i2++)
        y[i2][i1] = y[i2][i1-1] + k2[i2];

#endif

#ifdef __RK_3RD__

...

#endif

#ifdef __RK_4TH__

...

#endif
    }

  // *****

  // Ausgabe.

  for(i1 = 0; i1 <= num_steps; i1++)
    {
      double t = i1 * tau;
      printf("%9.6lf %9.6lf %9.6lf\n", t, y[0][i1], y[0][i1]-cos(t));
    }

  // *****

  return EXIT_SUCCESS;
}
```