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


// RK_dynamical_stepsize.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 anharmonischen Oszillator mit Potential
//
//   V(x) = \alpha x^n
//
// Mit dynamischer Schrittweitenanpassung.


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


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


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


// ********************
// Physikalische Parameter, etc.
// ********************

// Anharmonischer Oszillator, V(x) = \alpha x^n,
//   y = (x , v)
//   f = (v , -\alpha n x^n-1) .

const int N = 2;  // Anzahl der Komponenten von y bzw. f.


// const int n = 2;
// const double alpha = 0.5;
const int n = 20;
const double alpha = 1.0;

double y_0[N] = { 1.0 , 0.0 };  // Anfangsbedingungen y(t=0).


// Berechnet f(y(t),t) * tau.

void f_times_tau(double *y_t, double t, double *f_times_tau_, double tau)
{
  if(N != 2)
    {
      fprintf(stderr, "Error: N != 2!\n");
      exit(EXIT_FAILURE);
    }

  f_times_tau_[0] = y_t[1] * tau;
  f_times_tau_[1] = -alpha * ((double)n) * pow(y_t[0], ((double)(n-1))) * tau;
}


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


// ********************
// RK-Parameter.
// ********************


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

#ifdef __EULER__
const int order = 1;
#endif

#ifdef __RK_2ND__
const int order = 2;
#endif

#ifdef __RK_3RD__
const int order = 3;
#endif

#ifdef __RK_4TH__
const int order = 4;
#endif

// Maximale Anzahl der RK-Schritte.
const int num_steps_max = 10000;

// Sobald t >= t_max wird abgebrochen.
const double t_max = 10.0;

// Maximal zulässiger Fehler pro Schritt.
const double delta_abs_max = 0.001;

double tau = 1.0;  // Initiale (grobe) Schrittweite.


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


double t[num_steps_max+1];  // Diskretisierte Zeitachse.
double y[num_steps_max+1][N];  // Diskretisierte "Bahnkurven".


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


#ifdef __EULER__

...

#endif


#ifdef __RK_2ND__

// RK-Schritt (2nd order) um Schrittweite tau.

void RK_step(double *y_t, double t, double *y_t_plus_tau, double tau)
```

```
{
  int i1;

  // *****

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

  double k1[N];
  f_times_tau(y_t, t, k1, tau);

  // *****

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

  double y_[N];

  for(i1 = 0; i1 < N; i1++)
    y_[i1] = y_t[i1] + 0.5*k1[i1];

  double k2[N];
  f_times_tau(y_, t + 0.5*tau, k2, tau);

  // *****

  for(i1 = 0; i1 < N; i1++)
    y_t_plus_tau[i1] = y_t[i1] + k2[i1];
}

#endif



#ifdef __RK_3RD__

...

#endif



#ifdef __RK_4TH__

...

#endif



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



int main(int argc, char **argv)
{
  double d1;
  int i1, i2;

  // *****

  // Initialisiere "Bahnkurven" mit Anfangsbedingungen.

  t[0] = 0.0;

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

  // *****

  // Führe Euler/RK-Schritte aus.

  for(i1 = 0; i1 < num_steps_max; i1++)
    {
      if(t[i1] >= t_max)
        break;

      // *****

      // RK-Schritte.

      double y_tau[N], y_tmp[N], y_2_x_tau_over_2[N];

      // y(t)   --> \tau   y_{\tau}(t+\tau)
      RK_step(y[i1], t[i1], y_tau, tau);

      // y(t)   --> \tau/2   --> \tau_2   y_{2 * \tau / 2}(t+\tau)
      RK_step(y[i1], t[i1], y_tmp, 0.5*tau);
      RK_step(y_tmp, t[i1]+0.5*tau, y_2_x_tau_over_2, 0.5*tau);

      // *****

      // Fehlerabschätzung.

      double delta_abs = fabs(y_2_x_tau_over_2[0] - y_tau[0]);

      for(i2 = 1; i2 < N; i2++)
        {
          d1 = fabs(y_2_x_tau_over_2[i2] - y_tau[i2]);

          if(d1 > delta_abs)
            delta_abs = d1;
        }

      delta_abs /= pow(2.0, (double)order) - 1.0;

      // *****

      // Schrittweitenanpassung (maximale Veränderung um Faktor 5.0).

      d1 = 0.9 * pow(delta_abs_max / delta_abs, 1.0 / (((double)order)+1.0));

      if(d1 < 0.2)
        d1 = 0.2;

      if(d1 > 5.0)
        d1 = 5.0;

      double tau_new = d1 * tau;

      // *****

      if(delta_abs <= delta_abs_max)
        {
          // Akzeptieren des RK-Schrittes.

          for(i2 = 0; i2 < N; i2++)
            y[i1+1][i2] = y_2_x_tau_over_2[i2];

          t[i1+1] = t[i1] + tau;

          tau = tau_new;
        }
      else
        // Wiederholen des RK-Schrittes mit kleinerer Schrittweite.
        {
          tau = tau_new;

          i1--;
        }
    }

  int num_steps = i1;

```

```
285     // *****
286
287     // Ausgabe.
288
289     for(i1 = 0; i1 <= num_steps; i1++)
290       {
291         printf("%9.6lf %9.6lf\n", t[i1], y[i1][0]);
292       }
293
294     // *****
295
296     return EXIT_SUCCESS;
297   }
298
299
300
301   // **********
```