```
//  file: diffeq_routines.cpp
//
//  Routines for Euler's and 4th order Runge-Kutta diff. eq. routines
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//   30-Jan-2004 --- original version, translated from diffeq_routines.c,
//                    which was based on rk4.c from
//                    "Computational Physics" by Landau and Paez
//   30-Jan-2005 --- comments improved and function names changed
//   22-Jan-2006 --- made i local to loops
//
//   * Based originally on the discussion of differential equations
//      in Chap. 9 of "Computational Physics" by Landau and Paez
//   * See the 780.20 Session 6 background notes for formulas
//   * As a convention (advocated in "Practical C++"), we'll append
//       "_ptr" to all pointers.
//
//  To do:
//
//*************************************************************************

// include files
#include <iostream>              // note that .h is omitted
#include <iomanip>               // note that .h is omitted
#include <fstream>               // note that .h is omitted
#include <cmath>
#include "diffeq_routines.h"     // diffeq routine prototypes

const int NMAX=5;

//*************************************************************************
//
//   Euler's Algorithm Differential Equation Solver
//
// This routine takes all of the y's one step, from t to t+h.
//  The original values of y[0], y[1], etc. are lost.
//
// inputs:
//   N --- number of y(t)'s
//   t --- independent variable
//   y[] --- vector of y(t)'s
//   h --- step size
//   f --- function for the right hand sides
//   *params_ptr --- pointer to parameters for rhs function f
//
// outputs:
//   y[] --- predictions for the values of y(t+h)
//
//
//*************************************************************************
int
euler (const int N, double t, double y[], double h,
       double (*f) (double t, double y[], int i, void *params_ptr),
       void *params_ptr)
{
  for (int i = 0; i < N; i++)
  {
    y[i] += h * f (t, y, i, params_ptr)   // Eq.(6.45) in Session 6 notes
  }

  return (0);                   // successful completion
}
```



```
//*************************************************************************
//
//   4th Order Runge-Kutta Differential Equation Solver
//
// This routine takes all of the y's one step, from t to t+h.
//  The original values of y[0], y[1], etc. are lost.
//
// inputs:
//   N --- number of y(t)'s
//   t --- independent variable
//   y[] --- vector of y(t)'s
//   h --- step size
//   f --- function for the right hand sides
//   *params_ptr --- pointer to parameters for rhs function f
//
// outputs:
//   y[] --- predictions for the values of y(t+h)
//
// Notes:
//   * The algorithm is from Eqs. (6.47)-(6.48) in the Session 6 notes.
//
//*************************************************************************
int
runge4 (const int N, double t, double y[], double h,
        double (*f) (double t, double y[], int i, void *params_ptr),
        void *params_ptr)
{
  double y1[NMAX], y2[NMAX], y3[NMAX];  // intermediate y values
  double k1[NMAX], k2[NMAX], k3[NMAX], k4[NMAX]; // Runge-Kutta notation

  for (int i = 0; i < N; i++)
  {
    k1[i] = h * f (t, y, i, params_ptr);
    y1[i] = y[i] + k1[i] / 2.;  // argument for k2
  }

  for (int i = 0; i < N; i++)
  {
    k2[i] = h * f (t + h / 2., y1, i, params_ptr);
    y2[i] = y[i] + k2[i] / 2.;  // argument for k3
  }

  for (int i = 0; i < N; i++)
  {
    k3[i] = h * f (t + h / 2., y2, i, params_ptr);
    y3[i] = y[i] + k3[i];       // argument for k4
  }

  for (int i = 0; i < N; i++)
  {
    k4[i] = h * f (t + h, y3, i, params_ptr);
  }

  for (int i = 0; i < N; i++)
  {
    y[i] += (k1[i] + 2. * k2[i] + 2. * k3[i] + k4[i]) / 6.0;
  }

  return (0);                   // successful completion
}
```