```
#ifndef RK4_SOLVER_HH
#define RK4_SOLVER_HH

#include <iostream>
#include <cassert>
#include <vector>
//#define N 1250

class rk4 {
  public:
    bool enable_BH;
    double AA, BB, J, K, h_step, last_h_step, Atol, Rtol, fac, facmin, facmax, T_fin
al,
           dense_stpsze, barnes_theta, minx, maxx, miny, maxy, ox, oy, osint, ocost,
 oid;
    // cidx: Current Index
    // fidx: Futur Index
    int N, n_intvls, not_found, location, cidx, fidx, N_child, step_counter, SIZE_LI
ST;
    double *A, *B, *C, *sc, *x0, *y0, *theta0, *vx0, *vy0, *omega0, *x1, *y1, *theta
1,
           *x1h, *y1h, *theta1h, *vx1, *vy1, *omega1, *sc_x, *sc_y, *sc_theta, *bnod
es_idx,
           *xlim, *ylim, *xlim_next, *ylim_next;
    std::vector<double> barnes_list;
    std::vector<double> bnodes;

    rk4(int N_, double hi_step, double tol, double J_, double K_, int n_intvls_, dou
ble barnes_theta_, bool enable_BH_){
      N = N_; enable_BH = enable_BH_; AA = 1.; BB = 1.; J  = J_; K  = K_;
      step_counter = 0; n_intvls = n_intvls_; not_found = 1; barnes_theta = barnes_t
heta_;
      SIZE_LIST = 11; x0 = new double[N]; y0 = new double[N]; theta0 = new double[N]
;
      vx0 = new double [N]; vy0 = new double [N]; omega0 = new double [N];
      x1 = new double[N]; y1 = new double[N]; theta1 = new double[N];
      x1h = new double[N]; y1h = new double[N]; theta1h = new double[N];
      sc_x = new double[N]; sc_y = new double[N]; sc_theta = new double[N];
      xlim = new double[3]; ylim = new double[3]; xlim_next = new double[3]; ylim_ne
xt = new double[3];
      bnodes_idx = new double[N]; Rtol = tol; Atol = tol; fac = 0.9; facmax = 3.; fa
cmin = 1./3.;
      h_step = hi_step; last_h_step = hi_step; A = new double[10]; B = new double[9]
; C = new double[5];
      // Constructing the Butcher tables
      for(int i=0; i<10; i++){
        switch(i){
          case 0: A[i]=1./3.;
                  B[i]=1./8.;
                  C[i]=0.;
                  break;
          case 1: A[i]=-1./3.;
                  B[i]=3./8.;
                  C[i]=1./3.;
                  break;
          case 2: A[i]=1.;
                  B[i]=3./8.;
                  C[i]=2./3.;
                  break;
          case 3: A[i]=1.;
                  B[i]=1./8.;
                  C[i]=1.;
                  break;
          case 4: A[i]=-1.;
                  B[i]=1./12.;
                  C[i]=1.;
                  break;
          case 5: A[i]=1.;
                  B[i]=1./2.;
                  break;
          case 6: A[i]=1./8.;
                  B[i]=1./4.;
                  break;
```

```
          case 7: A[i]=3./8.;
                  B[i]=0.;
                  break;
          case 8: A[i]=3./8.;
                  B[i]=1./6.;
                  break;
          case 9: A[i]=1./8.;
                  break;
          default: break;
        }
      }
    };
    void initialize();
    void compute_solution(double T_final_);
    void compute_xx(double t_, double* x_, double* y_, double* theta_, double* outpu
tX, double* outputY, double* output_theta);
    void compute_Gs(double t, double* Gs_x, double* ff_x, double* Gs_y, double* ff_y
, double* Gs_theta, double* ff_theta);
    void compute_y1y1h(double t, double* Gs_x, double* ff_x, double* Gs_y, double* f
f_y, double* Gs_theta, double* ff_theta);
    void dense_output(double t_);
    void hermite(double actual_t, double myTheta, char* filenameDense);
    void nextStep();

    void zap(double* myArray){
      {assert(myArray!=NULL);}
      delete [] myArray;
      myArray = NULL;
    };
    void terminate(){
      zap(x0); zap(y0); zap(theta0); zap(x1); zap(y1); zap(theta1);
      zap(x1h); zap(y1h); zap(theta1h); zap(vx0); zap(vy0); zap(omega0);
      zap(sc_x); zap(sc_y); zap(sc_theta); zap(A); zap(B); zap(C);
      zap(xlim); zap(ylim); zap(xlim_next); zap(ylim_next);zap(bnodes_idx);
    };
    void barnes_compute(int cidx_, int &i, double xi, double yi, double thi,
                        double &sumx, double &sumy, double &sumtheta, int &N_comp,
double lgth);
    void smart_compute_xx(double t_, double* x_, double* y_, double* theta_,
                          double* outputX, double* outputY, double* output_theta );
    inline void init_lims();
    void find_square(double x, double y, bool nlim);
    inline void push_node(int i, double x_, double y_, double sint_, double cost_);


};

#endif
```