```c
#include "ode.h"
void euler_step(real t, pvector yt, ode_func f, real delta, pvector yt1, void *data)
{
    uint dim = yt->dim;

    uint i;

    assert(dim == yt1->dim);

    // evaluate function at point 't', current vector 'yt' into 'yt1'
    // compute the right hand side function and store the value in the vector yt1 ---> Phi
    f(t, yt, yt1, data);

    // update new vector 'yt1'
    for (i = 0; i < dim; i++)
    {
        yt->x[i] = yt->x[i] + delta * yt1->x[i];
    }
}

void runge_step(real t, pvector yt, ode_func f, real delta, pvector yt1, void *data)
{
    uint dim = yt->dim;

    pvector ym;
    uint i;

    assert(dim == yt1->dim);

    ym = new_vector(dim);

    // evaluate function at point 't', current vector 'yt' into 'yt1'
    f(t, yt, yt1, data);

    // update new vector 'ym'
    for (i = 0; i < dim; i++)
    {
        ym->x[i] = yt->x[i] + 0.5 * delta * yt1->x[i];
    }

    // evaluate function at point 't+delta/2', current vector 'ym' into 'yt1'
    f(t, ym, yt1, data);

    // update new vector 'yt'
    for (i = 0; i < dim; i++)
    {
        yt->x[i] = yt->x[i] + delta * yt1->x[i];
    }

    del_vector(ym);
}

void leapfrog_step(real t, pvector yt, ode_func f, real delta, pvector yt1, void *data)
{
    uint dim = yt->dim;
    real c_mass_spring = *((real *)data);

    (void)f;
    (void)yt1;

    assert(dim == yt1->dim);
    assert(dim == 2);

    yt->x[0] = yt->x[0] + 2.0 * delta * yt->x[1];
    yt->x[1] = yt->x[1] - 2.0 * delta * c_mass_spring * yt->x[0];
}

void crank_nicolson_step(real t, pvector yt, ode_func f, real delta, pvector yt1, void *data)
{
    uint dim = yt->dim;
    real c_mass_spring = *((real *)data);

    uint i;
    real onepdelta, onemdelta;

    (void)f;

    assert(dim == yt1->dim);
    assert(dim == 2);

    onepdelta = 1.0 + 0.25 * delta * delta * c_mass_spring;
    onemdelta = 1.0 - 0.25 * delta * delta * c_mass_spring;

    yt1->x[0] = onemdelta * yt->x[0] + delta * yt->x[1];
    yt1->x[1] = onemdelta * yt->x[1] - delta * c_mass_spring * yt->x[0];

    yt->x[0] = yt1->x[0] / onepdelta;
    yt->x[1] = yt1->x[1] / onepdelta;
}

```