```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <array>
#include <map>

//Constant expressions appearing in the problem
constexpr size_t dimension = 2;   //dimension of the reduced 1st-order problem
constexpr double PI = 3.14159265359;    //value of PI
constexpr double rollnum = 0.226121014;  //my roll number

//Definition of data types in the problem
typedef std::array<double, dimension> state_type;   //data type definition for dependant variables - array of x_0, x_1, ... x_n
typedef std::map<double, state_type> solution;  //data type definition for storing the list of calculated values ((hash)map of time -> state)

//Class template for the Runge Kutta solver using Butcher tableau
template <class State_Type, size_t order> class explicit_rk {
    //data type definnitions for storing the Butcher tableau
    typedef std::array<double, order> butcher_coefficients;
    typedef std::array<std::array<double, order>, order> butcher_matrix;
private:
    //information about the Butcher tableau
    butcher_matrix a;
    butcher_coefficients b, c;
    //temporary variables for intermediate steps
    std::array<State_Type, order> k;
public:
    //Constructor - just copy the Butcher tableau
    explicit_rk(butcher_matrix A, std::array<double, order> B, std::array<double, order> C) : a(A), b(B), c(C) {
        k = {};    //zero-initialize k
    }

    //Destructor - nothong to do
    ~explicit_rk() {

    }

    //The stepper function, calculates x_{n+1} given the differential equation, x_{n}, t and step size
    void do_step(void (*Diff_Equation)(const State_Type& x, const double& t, State_Type& dxdt), State_Type& x, const double& t, const double& dt){
        State_Type result = x;  //temporary variable for storing the result

```

```
        //loops for evaluating k1, k2 ... k_n
        for (size_t i = 0; i < order; i++) {
            State_Type sum{}, dxdt; //temporary variables for k's and the derivatives
            for (size_t j = 0; j < i; j++) {
                sum = sum + dt * a[i][j] * k[j];    //compute a_{ij} * k_j
            }
            sum = x + sum;  //compute x_{n} + a_{ij} * k_j
            Diff_Equation(sum, t + c[i] * dt, dxdt);    //evaluate dx/dt at (x_{n} + a_{ij} * k_j, t_{n} + c_{i} * dt) according to Runge Kutta
            k[i] = dxdt;    //store the dx/dt as k_i
        }

        //loop for calculating x_{n+1} using the k's
        for (size_t i = 0; i < order; i++) {
            result = result + dt * b[i] * k[i]; //weighted average of k's with b's as weights
        }

        //return the result
        x = result;
    }
};

//Overload the + operator to be able to add two vectors
state_type operator + (state_type const &x, state_type const &y) {
    state_type z;
    for (size_t i = 0; i < dimension; i++) {
        z[i] = x[i] + y[i]; //add the individual components and store in z
    }
    return z;   //return the resulting vector z
}

//Overload the * operator to be able to multiply numbers and vectors
state_type operator * (double const &a, state_type const &x) {
    state_type z;
    for (size_t i = 0; i < dimension; i++) {
        z[i] = a * x[i];    //multiply the individual components and store in z
    }
    return z;   //return the resulting vector z
}

//This is the differential Equation, reduced to first-order
void Pendulum(const state_type& x, const double& t, state_type& dxdt){
    dxdt[0] = x[1];
    dxdt[1] = -4.0 * PI * PI * sin(x[0]);
}
```

```cpp

int main(){
    //Using the class template, creates a class object for the Runge Kutta solver with a given butcher tableau
    explicit_rk <state_type, 4> rk4_stepper({0,0,0,0,   //Butcher a matrix
                                             .5,0,0,0,
                                             0,.5,0,0,
                                             0,0,1,0},
                                             {1.0/6.0 , 1.0/3.0 , 1.0/3.0 , 1.0/6.0},    //Butcher b coefficiants
                                             { 0.0 , 0.5 , 0.5 , 1.0});  // Butcher c coefficients

    solution x_t;   //variable to store the calculations

    size_t STEPS = 1000;  //number of steps
    double t_0 = 0.0;   //initial time
    double t_1 = 1.0;   //final time
    double dt = (t_1 - t_0) / (STEPS - 1); //step size
    state_type x = {0.0, rollnum};   //initial values for dependant variables

    //Step through the domain of the problem and store the solutions
    x_t[t_0] = x;   //store initial values
    for (size_t i = 0; i < STEPS; i++) {
        rk4_stepper.do_step(Pendulum, x, NULL, dt);    //step forward
        x_t[t_0 + i * dt] = x;  //store the calculation
    }

    std::ofstream outfile;  //file handle to save the results in a file
    outfile.open("tableau.txt", std::ios::out | std::ios::trunc );
    for (auto const& temp : x_t){
        outfile << temp.first << "\t" << temp.second[0] << "\t" << temp.second[1] << std::endl;
    }
    outfile.close();
}
```