```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <vector>
#include <fstream>

using namespace std;

// A C++ class definition for two point BVPs

class TwoPointBVP
{
protected:
        double * domain;
        bool leftBdryIsDirichlet; // value is "true" if left boundary is Dirichl
et
        bool rightBdryIsDirichlet; // value is "true" if right boundary is Diric
hlet
        double * leftBdryValues; // contains gamma_0 and g_0 (see course notes)
        double * rightBdryValues; // contains gamma_0 and g_0 (see course notes)
        bool reactionIsPresent; // value is "true" if there is a reaction
        bool forcingFunctIsPresent; // value is "true if there is a forcing funt
ion(ie not homogenous)
        bool trueSolIsPresent; // value is true if there is a true solution
        double(*diffusion) (vector<double> &); // diffusion coeff, k(x)
        double(*reaction) (vector<double> &); // reaction coeff, r(x)
        double(*partialreactionpartialu) (vector<double> &); //pd_r/pd_u evaled
at (x,u)
        double(*forcingFunct) (vector<double> &); // forcing function f(x)
        double(*trueSolu) (vector<double> &); // true solution
public:
        //Constructor of the class, it takes two input.
        TwoPointBVP(double *dom, double(*dFunc) (vector<double> &));

        // Set data for the left boundary.
        void set_left_bdry(bool _leftIsDirichlet, double *val);

        // Set data for the right boundary.
        void set_right_bdry(bool _rightIsDirichlet, double *val);

        // Set reaction
        void set_reaction(double(*rFunctOne) (vector<double> &),
                                          double(*rFunctTwo) (vector<double> &))
;

        // Set forcing function
        void set_forcing_function(double(*fFunct) (vector<double> &));

        // Set true solution
        void set_true_solution(double(*TrueSol)(vector<double> &));

        /********************
        Functions below can be called from outside TwoPointBVP to access various
        pertaining aspects of TwoPointBVP
        ********************/

        //Return domain (see above)
        double *get_domain() const;
        // Return leftBdryIsDirichlet
        bool left_bdry_is_Dirichlet() const;

        // Return rightBdryIsDirichlet
        bool right_bdry_is_Dirichlet() const;

        //Return left boundary values; ie gamma_0 & g0
        double * get_left_bdry_values() const;

        //Return right boundary values; ie gamma_L & g_l
        double * get_right_bdry_values() const;

        // Return if a reaction is present
        bool reaction_is_present() const;

        //Return if a forcing function is present
        bool forcing_fucntion_is_present() const;

        //Return  if a true solution is present
        bool true_solution_is_present() const;

        // Retrun the value of k(x)
        double eval_diffusion(vector<double> &x) const;

        // Return the value of r(x,u)
        vector<double> eval_reaction(vector<double> &par) const;


        //Return the value of f(x)
        double eval_forcing_function(vector<double> &x) const;

        //Return the value of the true Soln @ x
        double eval_true_solution(vector<double> &x) const;

        //Display info about the two point BVP
        void display_info_TwoPointBVP() const;

        // Return a vector with numEvals evenly spaced evaluations
        // of the true solution including and between domLeft and
        // domRight
        vector<double> true_solution(int numEvals, double domLeft, double domRig
ht);

        // TwoPointBVP Destructor
        ~TwoPointBVP();
};
```