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

using namespace std;

const double LEFT_BOUND = 0.0;
const double RIGHT_BOUND = 2.0;

const double STEP = 0.001;
const size_t SIZE = (RIGHT_BOUND - LEFT_BOUND) / STEP;

struct GasData
{
    GasData(double _P, double _U, double _RO, double _E, double _gamma) {
        P = _P;
        U = _U;
        RO = _RO;
        E = _E;
        gamma = _gamma;
    }
    double P, U, RO, E, gamma;
};

/*
    Function for setting initial data
    @param tasks - vector of structs GasData
*/
void init_data(double* P, double* P_new, double* U, double* U_new, double* RO, double*
RO_new,
    double* NU, double* NU_new, double* E, double* E_new, double* Q, double *X, double*
deltaS, vector<GasData> tasks)
{
    // create mesh
    for (size_t i = 0; i < SIZE; i++) {
        X[i] = STEP * i;
    }

    if (tasks.size() == 2) {
        GasData task_left = tasks[0];
        GasData task_right = tasks[1];

        for (size_t i = 0; i < SIZE; i++) {
            Q[i] = 0;
            if (i <= SIZE / 4 - 1) {
                RO[i] = task_left.RO;
                NU[i] = 1.0 / task_left.RO;
                P[i] = task_left.P;
                E[i] = task_left.E;
            }
            else {
                U[i] = task_right.U;
                RO[i] = task_right.RO;
                NU[i] = 1.0 / task_right.RO;
                P[i] = task_right.P;
                E[i] = task_right.E;
            }
        }

        for (size_t i = 0; i < SIZE; i++) {
            if (i <= SIZE / 4) {
                U[i] = task_left.U;
            }
            else {
                U[i] = task_right.U;
            }
        }
    }
    else if(tasks.size() == 1) {
        GasData task = tasks[0];

        for (size_t i = 0; i < SIZE; i++) {
            Q[i] = 0;
            U[i] = task.U;
            RO[i] = task.RO;
            NU[i] = 1.0 / task.RO;
            P[i] = task.P;
            E[i] = task.E;
        }
    }

    for (size_t i = 0; i < SIZE; i++) {
        deltaS[i] = STEP * RO[i];
    }
}

/*
    Function - one step of computing
*/
void compute_step(double* P, double* P_new, double* U, double* U_new, double* RO,
double* RO_new,
    double* E, double* E_new, double* Q, double* Q_new, double tau, double* NU, double*
NU_new, double* X, double* deltaS, double gamma)
{

    U[0] = 1;
    U[SIZE - 1] = 0;

    for (size_t k = 1; k < SIZE; k++) {
        U_new[k] = U[k] - tau * (P[k] + Q[k] - P[k - 1] - Q[k - 1]) / deltaS[k];
    }
    U_new[0] = 1;
    U_new[SIZE - 1] = 0;

    for (size_t k = 0; k < SIZE - 1; k++) {
        NU_new[k] = NU[k] + (tau * (U_new[k + 1] - U_new[k]) / deltaS[k]);
        RO_new[k] = 1.0 / NU_new[k];
    }

    for (size_t k = 0; k < SIZE - 1; k++) {
        E_new[k] = E[k] - (P[k] + Q[k]) * (NU_new[k] - NU[k]);
    }

    //P[0] = 1;
    for (size_t k = 0; k < SIZE - 1; k++) {
        P_new[k] = (gamma - 1) * E_new[k] / NU_new[k];
    }
    //P_new[0] = 1;

    for (size_t k = 0; k < SIZE - 1; k++) {
        double C = 0;
        if ((U_new[k + 1] - U_new[k]) < 0)
        {
            // Q from my schema
            C = pow(gamma * P_new[k] / RO_new[k], 0.5);
            double tmp1 = fabs(U_new[k + 1] - U_new[k]) / NU_new[k];
            double tmp2 = (gamma + 1) * fabs(U_new[k + 1] - U_new[k]) / 4;
            Q_new[k] = tmp1 * (tmp2 + sqrt(pow(tmp2, 2) + pow(C, 2)));
        }
        else Q_new[k] = 0;
    }
    for (size_t i = 0; i < SIZE; i++)
    {
        X[i] = X[i] + tau * U_new[i];
    }
}

/*
*   Function for making new tau
*/
double compute_next_tau(double gamma, double *P, double *RO, double *deltaS, double
*tau_k, double *tau_uv, double *U_new, double tau) {
    double C = 0.0;

    for (size_t i = 0; i < SIZE - 1; i++) {
        C = pow(gamma * P[i] / RO[i], 0.5);

        tau_k[i] = deltaS[i] / (C * RO[i]);

        if ((U_new[i] - U_new[i]) != 0) {
            tau_uv[i] = 1.0 / (8 * fabs(U_new[i] - U_new[i]));
        }
        else
            tau_uv[i] = 10;
    }

    double min_k = tau_k[0];
    double min_uv = tau_uv[0];

    for (size_t p = 0; p < SIZE - 1; p++) {
        if (tau_k[p] <= min_k) {
            min_k = tau_k[p];
        }

        if (tau_uv[p] <= min_uv) {
            min_uv = tau_uv[p];
        }
        if (min_uv < min_k) {
            min_k = min_uv;
        }
    }
    if (min_k <= 1.2 * tau) return min_k;
    else return 1.2*tau;
    //return *min_element(taus.begin(), taus.end());
}


void calc()
{
    /*
        Block of declaration variables
    */
    double *U, *E, *RO, *P, *Q, *C, *NU,
        *U_new, *E_new, *RO_new, *C_new, *P_new, *Q_new, *NU_new;
    double *X, *deltaS;

    double tau = 0.00005,
        END_TIME = 0.75,
        current_time = 0.0;


    double* tau_k = new double[SIZE];
    double* tau_uv = new double[SIZE];

    /*
        Allocate memory
    */
    P = new double[SIZE];
    P_new = new double[SIZE];
    U = new double[SIZE];
    U_new = new double[SIZE];
    RO = new double[SIZE];
    RO_new = new double[SIZE];
    NU = new double[SIZE];
    NU_new = new double[SIZE];
    E = new double[SIZE];
    E_new = new double[SIZE];
    Q = new double[SIZE];
    Q_new = new double[SIZE];
    X = new double[SIZE];
    deltaS = new double[SIZE];

    /*
        Definition of tasks
    */
    GasData task1_left(4.0 / 3.0, 1.0, 4.0, 0.5, 5.0 / 3.0);
    GasData task1_right(0.0002 / 3.0, 0.0, 1.0, 0.0001, 5.0 / 3.0);

    GasData task2(20.0 / 7.0, 0.0, 4.0, 25.0 / 14.0, 7.0 / 5.0);

    GasData task3_left(7.59375, 0.0, 12.65625, 0.9, 5.0 / 3.0);
    GasData task3_right(2.0 / 7.0, 0.0, 5.0 / 14.0, 1.2, 5.0 / 3.0);

    init_data(P, P_new, U, U_new, RO, RO_new, NU, NU_new, E, E_new, Q, X, deltaS, {
task1_left, task1_right });


    ofstream fout1("test_1.txt");
    ofstream fout_time("tau_from_time.txt");
    fout_time << "time\t" << "tau" << endl;

    fout1 << "time\t" << "massa\t" << "impuls\t" << "Ekin\t" << "Evn\t" << "Epoln\t" <<
endl;

    while (current_time < END_TIME) {

        double C = 0.0;

        for (size_t i = 0; i < SIZE - 1; i++) {
            C = pow((task1_left.gamma * P[i] / RO[i]), 0.5);

            if (P[i] < 0)
            {
                cout << "DAVLENIE SUKA PREDATEL!" << endl;
                cout << "P[i]: " << P[i] << endl;
                break;
            }
            if (RO[i] < 0)
                cout << "PLOTNOST SUKA PREDATEL!" << endl;

            tau_k[i] = deltaS[i] / (C * RO[i]);

            if ((U[i + 1] - U[i]) != 0) {
                tau_uv[i] = 1.0 / (8 * fabs(U[i + 1] - U[i]));
            }
            else
                tau_uv[i] = 10;
        }

        double min_k = tau_k[0];
        double min_uv = tau_uv[0];

        for (size_t p = 0; p < SIZE - 1; p++) {
            if (tau_k[p] <= min_k) {
                min_k = tau_k[p];
            }

            if (tau_uv[p] <= min_uv) {
                min_uv = tau_uv[p];
            }
            if (min_uv < min_k) {
                min_k = min_uv;
            }
        }
        //if (min_k <= 1.2 * tau) tau = min_k;
        //else tau = 1.2 * tau;



        compute_step(P, P_new, U, U_new, RO, RO_new, E, E_new, Q, Q_new, tau, NU,
NU_new, X, deltaS, task1_left.gamma);
        current_time += tau;

        //tau = compute_next_tau(task1_left.gamma, P, RO, deltaS, tau_k, tau_uv, U_new,
tau);
        //fout_time << current_time << "\t" << tau << endl;

        // Rewrite arrays from layer n+1 to layer n
        for (int k = 1; k < SIZE - 1; k++)
        {
            U[k] = U_new[k];
            E[k] = E_new[k];
            RO[k] = RO_new[k];
            NU[k] = NU_new[k];
            P[k] = P_new[k];
            Q[k] = Q_new[k];
        }
        cout << "tau: " << tau << endl;

        double MV = 0, Ek = 0, Ev = 0, E_full = 0, Massa = 0;
        for (int k = 0; k < SIZE-1; k++)
        {
            MV += U_new[k] * deltaS[k];
            Ek += deltaS[k] * (U_new[k] * U_new[k] * 0.25 + U_new[k + 1] * U_new[k + 1]
* 0.25);
            Ev += E_new[k] * deltaS[k];
            Massa += deltaS[k];
        }
        E_full += Ek + Ev;
        fout1 << current_time << "\t" << Massa << "\t" << MV << "\t" << Ek << "\t" <<
Ev << "\t" << E_full << endl;
    }

    ofstream fout("task_1.txt");
    fout << "X:" << "\t" << "U:" << "\t" << "P:" << "\t" << "RO:" << "\t" << "E:" <<
"\t" << "Q:" << endl;


    for (size_t j = 1; j < SIZE; j++)
    {
        fout << X[j] << "\t" << U[j] << "\t" << P[j] << "\t" << RO[j] << "\t" << E[j]
<< "\t" << Q[j] << endl;
    }
    fout.close();
    fout1.close();
    fout_time.close();

    /*
        Free memory
    */
    delete[]P; delete[]P_new;
    delete[]U; delete[]U_new;
    delete[]RO; delete[]RO_new;
    delete[]NU; delete[]NU_new;
    delete[]E; delete[]E_new;
    delete[]Q; delete[]X; delete[]deltaS;
}


int main()
{
    calc();
    return 0;
}
```