#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;

}