```cpp
#include <iostream>
#include <armadillo>
#include <fstream>
#include "functions.h"
#include "pdesolver.h"
#include "time.h"

using namespace std;
using namespace arma;

void analytical(mat *U);

int main()
{
    int Nx, Nt; double T, dt, h; mat u, u2, u3;

    //-------------------------------------------------------------------------------------
    // Solving 2+1 dimensional diffusion equation with implicit, explicit and analytic method
    //-------------------------------------------------------------------------------------

    initialize(&Nx,&Nt,&T);            // Reading in Nx, Nt and T from user

    h = 1./Nx; dt = T / Nt;            // Spacial steplength h and time step dt
    u = zeros<mat>(Nx,Nx);             // u is a function showing concentration for a given x and y. u(x,y)

    for (int i = 0; i<Nx; i++){        // Setting boundary condition u(0,y) = 1. All other boundarys are 0.
        u(0,i) = 1;
    }
    u2 = u; u3 = zeros<mat>(Nx,Nx);       // u2 will be solved by explicit method, u3 Laplace,

    BackwardEuler(&u, Nx, h, dt);      // Solving as a function of h and dt
    ForwardEuler(&u2, Nx, Nt, h, dt);  // Solving as a function of h and dt
    analytical(&u3);

    ofstream myfile, myfile1, myfile2, myfile3;
    myfile.open("Implicit.txt"); myfile2.open("Explicit.txt"); myfile3.open("Analytic.txt");
    myfile << u; myfile2 << u2; myfile3 << u3;
    myfile.close(); myfile2.close(); myfile3.close();

    //-------------------------------------------------------------------------------------------------------
    // Solving 1+1 dimensional diffusion equation using Markov Chain, explicit, implicit and analytical solver.
    //-------------------------------------------------------------------------------------------------------

    int N0; vec X, X1, X2, X3; // N0 is number of particles at x=0. X is the position vector for all particles
    N0 = 1000; X = zeros<vec>(N0); //All N0 particles start at 0, so X has length N0 with values 0.
    X3 = X;
    X2 = zeros<vec>(Nx);           // Setting initial condition. This is due to the fact that v = u - us
    X2(0) = 0; X2(Nx-1) = 0;
    for (int i=1;i<Nx-1;i++){
        X2(i) = i*h - 1;
    }

    clock_t start, start1, start2, finish, finish1,finish2; double t0,t1,t2;
    PDESolver D1(Nt,Nx,dt,h);
    start = clock();
    D1.RandomWalk(&X);
```

```
    finish = clock();
    D1.Analytical(&X1);
    start1 = clock();
    D1.CrankNicolson(&X2);
    finish1 = clock();
    start2 = clock();
    D1.GaussRandomWalk(&X3);
    finish2 = clock();

    t0 = ((finish - start)/CLOCKS_PER_SEC); t1 = ((finish1 - start1)/CLOCKS_PER_SEC); t2
= ((finish2 - start2)/CLOCKS_PER_SEC);
    cout << "Random walk used " << t0 << "seconds" << endl;
    cout << "Crank Nicolson used " << t1 << "seconds" << endl;
    cout << "Random walk - Gauss - used " << t2 << "seconds" << endl;

    myfile.open("RandomWalk_a.txt"); myfile1.open("Analytical_1d.txt");
myfile2.open("CrankNicolson.txt"); myfile3.open("RandomWalk_b.txt");
    myfile << X; myfile1 << X1; myfile2 << X2; myfile3 << X3;
    myfile.close(); myfile1.close(); myfile2.close(); myfile3.close();

    return 0;
}
```