```
// REKESH ALI
// M475 TEAM F
// Conservation PDE

#include <stdio.h>
#include <math.h>
#include "global.h" // includes all subroutines
#include "declarations.h"  // to get rid of compiler warnings

int main(int argc, char * argv[]){
        //============================ Initialize I/O
        char inp[20];
        if(argc == 1){
                sprintf(inp, "input");
        }else{
                sprintf(inp, "%s.i", argv[1]);
        }

        //============================ READ INPUTS
        int MM;
        double tend, dtout, factor;
        readfile(inp, &factor, &dtout, &tend, &MM); // reads from
filename inp
        //============================ CREATE MESH
        M = (double)MM*(b-a); // number of CV's
        M = (int)M;
        const int W = M;
        dx = 1./((double)(MM)), dy = 1./((double)(MM)); // spacing
between nodes
        double X[M+2], Y[M+2]; // nodal array
        mesh(X, Y); // fills array with positions of nodes

        //============================ SET TIMESTEP
        double t0 = 0.0; // start time
        double kmax = fmax(kl, ks);
        double Cmin = fmin(Cl, Cs);
        double dtEXPL = dx*dx*rho*Cmin/(4.*kmax);
        dt = factor*dtEXPL; // dt fraction of CFL for stability
purposes
        int Nend = (int)((tend - t0)/dt) + 1; // number of timesteps
        double Nend2 = ((tend - t0)/dt) + 1;
        int nsteps = 0; // initialize timestep
        double time = t0; // initialize time
        double tout = fmax(dtout, dt); // time for printing to fil

        printf("M = %i, tend = %f, dt = %.15e, Nend = %i\n", M, tend,
dt, Nend);

        //============================ INITIALIZE PROFILE
        double T[M+2][M+2], E[M+1][M+1], p[M+1][M+1]; // solution
```

```
array and max error
        init(W, T, E, p); // fills solution array
        //=========================== BEGIN TIMESTEPPING
        double Fx[M+1][M+1], Fy[M+1][M+1]; // initialize flux array
        double F0[M+1]; // flux distribution array at boundary
        BCFlux(F0); // gaussian vs uniform distribution
        nbar = 1;
        maxwidth = 0;
        output(W, X, Y, T, Fx, Fy, E, p, time, nsteps, tend); // print
to file

 //   #pragma omp parallel for num_threads(4) schedule(dynamic)  //
parallel for loop
        for(nsteps = 1; nsteps <= Nend; nsteps++){
                time = nsteps*dt; // current time
                flux(W, Fx, Fy, T, p, F0); // current flux at walls
                pde(W, E, Fx, Fy); // updates energy with forward
euler
                eos(W, E, T, p, Fx, Fy); // updates temperatures and
phases
                if(time > tout){ // when time to print
                        output(W, X, Y, T, Fx, Fy, E, p, time,
nsteps, tend); // print to file
                        tout = tout + dtout; // next print time
        }
                if(maxwidth){
                break;}
        }
}
```