```
 //const int * W;
 int M, BCType, FType, nbar, maxwidth;
 double a, b;
 double dt, dx, dy;
 double Cs, Cl;
 double ks, kl;
 double rho, L;
 double h, Tinf;
 double T0, Q0, Tm;
#define pi 3.14159265358979323846
```

```
function animate(inputfile, outputfile, x, y, MM, Q, dtout, dtquit)
    gifname = sprintf('%s_%s_%d_%d.gif', inputfile, outputfile, MM,
Q); % name of gif
    command = sprintf('rm %s', gifname); % deletes old
    system(command); % does above
    [O, Omin, Omax, N] = getoutput(outputfile, 0); % gets min/max Temp
for scale

    h = figure;
    hold on
    set(gcf, 'Units', 'Normalized', 'OuterPosition', [0.1, 0.1, .6,
0.8]);
    set(gcf,'color','w');

    for i = 1:N
        surf(x,y,O(:,:,i));
        if(i == 1)
            if(outputfile(1:4) == 'temp') % no colorbar if phase
                c = colorbar;
                ylc = ylabel(c, 'Temperature (K)', 'FontSize', 20,
'Rotation', 270);
                posy = get(ylc, 'Position');
                set(ylc, 'Position', posy + [2, (Omax-Omin)/2 -
posy(2), 0]);
                caxis([Omin Omax]);
                ylabel('Y Dimension (cm)');
                xlabel('X Dimension (cm)');
            elseif(outputfile(1:5) == 'phase')
                ylabel('Y Dimension (cm)');
                xlabel('X Dimension (cm)');
            end
        end

        titstr = sprintf('time elapsed = %f s', (i-1)*dtout);
        if(outputfile(1:4) == 'temp') % no colorbar if phase
            titstr = sprintf('Temperature, %s', titstr);
            title(titstr);
        elseif(outputfile(1:5) == 'phase')
            titstr = sprintf('Phase, %s', titstr);
            title(titstr)
        end

        set(gca, 'FontSize', 20);
        colormap jet
        shading interp;
        clear figure
        view(0,90);
%         hold on;
        %pause(dtout/scale); % time accurate plot
        gifmaker(gifname, h, i); % saves gif
        if((i-1)*dtout > dtquit) % break at steady state
            break;
        end

    end
end
```

```
function data = fgetmat(fid)
% Rekesh Ali
% Function quits at eof or empty line!
i = 1;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line)
        break;
    end
    data(i,:) = sscanf(line, '%f');
        i = i + 1;
end
```

```
 filedir = sprintf("../outputs/flxdst.o"); % output file dir
    fid = fopen(filedir); % opens file
    picname = 'flxdst8000W1000MM.png';

    data = fgetmat(fid);
    fclose(fid);
    x = data(:,1);
    flux = data(:,2);

    fig = figure;
    hold on
    set(gcf, 'Units', 'Normalized', 'OuterPosition', [0.1, 0.1, .6,
0.8]);
    set(gcf,'color','w');
    plot(x*100, flux, 'LineWidth', 2);
%     xlim([0 max(time)]);
    xlabel('Position (cm)');
    ylabel('Flux (W)');
%     tit = sprintf('%s, Pool Size, MM = %.0f, Q = %.0fW', inputfile,
MM, Q);
    title('Boundary Flux Distribution, MM = 5000');
    set(gca, 'FontSize', 20);
    grid on

    frame = getframe(fig);
    im = frame2im(frame);
    [imind,cm] = rgb2ind(im,256);
    imwrite(imind,cm,picname,'png', 'WriteMode','overwrite');
```

```
function [O, Omin, Omax, i] = getoutput(outputfile, j)
    filedir = sprintf("../outputs/%s.o",outputfile); % output file dir
    fid = fopen(filedir); % opens file

    Omin = 1000000000000; % arbitrary initializers
    Omax = 0;
    i = 0;
    while ~feof(fid) % until the end of file
        i = i + 1;
        if j == 0 % animate
            O(:,:,i) = fgetmat(fid); % gets output matrix at every
time-step
            mintemp = min(min(O(:,:,i)));
            maxtemp = max(max(O(:,:,i)));
        else % just last one
            O = fgetmat(fid);
            mintemp = min(min(O));
            maxtemp = max(max(O));
        end


        if(maxtemp > Omax) % finds min and max Temps for plot scale
            Omax = maxtemp;
        end
        if(mintemp < Omin)
            Omin = mintemp;
        end
    end
    fclose(fid);
end
```

```
function gifmaker(gifname,figlabel,i)
    % Makes a gif, duh
    frame = getframe(figlabel);
    im = frame2im(frame);
    [imind,cm] = rgb2ind(im,256);
    % Write to the GIF File
    if i == 1
      imwrite(imind,cm,gifname,'gif', 'Loopcount',inf,
'WriteMode','overwrite');
    else
      imwrite(imind,cm,gifname,'gif','WriteMode','append');
    end
end
```

```
function plotend(inputfile, outputfile, x, y, MM, Q, tend)
    [O, Omin, Omax, ~] = getoutput(outputfile, 1);
    Omax
    Omin
    picname = sprintf('%s_%s_%d_%d.png', inputfile, outputfile, MM,
Q);
    %Omax = 7064.03;
    Omin = 298;
    fig = figure;
    hold on
    set(gcf, 'Units', 'Normalized', 'OuterPosition', [0.1, 0.1, .6,
0.8]);
    set(gcf,'color','w');

    surf(x,y,O);


    if(outputfile(1:4) == 'temp') % no colorbar if phase
        c = colorbar;
        ylc = ylabel(c, 'Temperature (K)', 'FontSize', 20, 'Rotation',
270);
        posy = get(ylc, 'Position');
        set(ylc, 'Position', posy + [2, (Omax-Omin)/2 - posy(2), 0]);
        caxis([Omin Omax]);
        ylabel('Y Dimension (cm)');
        xlabel('X Dimension (cm)');
    elseif(outputfile(1:5) == 'phase')
        ylabel('Y Dimension (cm)');
        xlabel('X Dimension (cm)');
    end

    if(outputfile(1:4) == 'temp') % no colorbar if phase
        titstr = sprintf('%s, Temperature, MM = %.0f, Q = %.0fW, Time
= %4.2fms', ...
            inputfile, MM , Q, 1000*tend);
        title(titstr);
    elseif(outputfile(1:5) == 'phase')
        titstr = sprintf('%s, Phase, MM = %.0f, Q = %.0fW, Time =
%4.2fms', ...
            inputfile, MM , Q, 1000*tend);
        title(titstr)
    end

    set(gca, 'FontSize', 20);
    colormap jet
    shading interp;
    clear figure
    view(0,90);
    frame = getframe(fig);
    im = frame2im(frame);
    [imind,cm] = rgb2ind(im,256);
    imwrite(imind,cm,picname,'png', 'WriteMode','overwrite');
end
```

```
function plotmain(inputfile,outputfile, dtquit, tend)
    % Example: plotmain('debug','temp') runs the temp output from the
debug
    % input file
    [x, y, dtout, ~, Tm, MM, Q] = readinput(inputfile); % gets mesh
and dtout
    plotpoolsize(inputfile, MM, Q);
    plotend(inputfile, outputfile, x, y, MM, Q, tend/1000);
%     animate(inputfile, outputfile, x, y, MM, Q, dtout, dtquit)
end
```

```
function plotpoolsize(inputfile, MM, Q)
    filedir = sprintf("../outputs/values.o"); % output file dir
    fid = fopen(filedir); % opens file
    picname = sprintf('%s_size_%d_%d.png', inputfile, MM, Q);

    fgetl(fid); % useless
    data = fgetmat(fid);
    fclose(fid);
    time = data(:,2);
    width = data(:,3);
    depth = data(:,4);
   % energy = data(:,5);

    fig = figure;
    hold on
    set(gcf, 'Units', 'Normalized', 'OuterPosition', [0.1, 0.1, .6,
0.8]);
    set(gcf,'color','w');
    plot(time,width, 'LineWidth', 2);
    plot(time,depth, 'LineWidth', 2);
    xlim([0 max(time)]);
    xlabel('Time (ms)');
    ylabel('Size (cm)');
    tit = sprintf('%s, Pool Size, MM = %.0f, Q = %.0fW', inputfile,
MM, Q);
    title(tit);
    legend({'Width','Depth'})
    set(gca, 'FontSize', 20);
    grid on

    frame = getframe(fig);
    im = frame2im(frame);
    [imind,cm] = rgb2ind(im,256);
    imwrite(imind,cm,picname,'png', 'WriteMode','overwrite');
end
```

```
function [x, y, dtout, tend, Tm, MM, Q] = readinput(inputfile)
    % Reads input file for automatic plot generation
    filename = sprintf('../inputs/%s.i', inputfile); % file directory
    fID = fopen(filename); % open file

    fgetl(fID); % useless data
    fgetl(fID);

    buff = fgetl(fID); % line with dtout
    data = strsplit(buff);
    dtout = str2double(data{2});% dtout secured
    tend = str2double(data{3}); % tend

    fgetl(fID); % more useless
    fgetl(fID);
    clear buff data
    buff = fgetl(fID); % line with MM, etc
    data = strsplit(buff);
    MM = str2double(data{1}); % dimensional data
    a = str2double(data{2});
    b = str2double(data{3});

    fgetl(fID); % more useless
    fgetl(fID);
    clear buff data
    buff = fgetl(fID); % line with Tm, etc
    data = strsplit(buff);
    Tm = str2double(data{3});

    fgetl(fID); % more useless
    fgetl(fID);
    fgetl(fID);
    fgetl(fID);
    clear buff data
    buff = fgetl(fID); % line with Tm, etc
    data = strsplit(buff);
    Q = str2double(data{2});


    M = MM*(b-a);
    gridline = linspace(a,b,M);
    [x,y] = meshgrid(gridline); % mesh for plotting secured
    x = x*100;
        y = y*100;
    fclose(fID); % being neat
end
```

```
#include "global.h"
#include <math.h>
#include <stdio.h>
void BCFlux(double F0[]){
        // creates a flux vector that holds value at each node
        // corresponding to gaussian distribution
        FILE * OUT;
        OUT = fopen("outputs/flxdst.o", "w"); // output flux 
distribution

        double x;
        double center = 0.5*(b-a); // location of peak
        double scale = 5/(b); // will look like gaussian of 5 max
        double stddev = 1/(scale*sqrt(2*pi)); // width of flux
        for(int i = 1; i <= M; i++){ // for every node
                x = a + ((double)i - 0.5)*dx; // location of current 
node
                if(!FType){ // Gaussian distribution
                        F0[i] = Q0*exp( -(x - center)*(x - center)/
(2*stddev*stddev)); // flux at node
                }else{ // Uniform distribution
                        F0[i] = Q0; // all flxues same
                }
                fprintf(OUT, "%.6f %11.4f\n", x, F0[i]); // keeping 
track
        }
        fclose(OUT);
}
```

```
//=================================== Compute conductivities
#include "global.h"
double conduct(double p){
    // check liquid fraction of node
    if(p == 0){
        return ks;
    }else if(p == 1){
        return kl;
    }else{
        double kinv =  p/kl + (1-p)/ks; // mushy
        return 1/kinv;
    }
}
```

```
// declare functions to get rid of compiler warnings
void readfile(char filename[], double *factor, double *dtout, double
*tend, int *MM);
void mesh(double X[], double Y[]);
void init(const int W, double T[][W+2], double E[][W+1], double p[]
[W+1]);
void BCFlux(double F0[]);
void output(const int W, double X[], double Y[], double T[][W+2],
double Fx[][W+1], double Fy[][W+1],
            double E[][W+1], double p[][W+1], double time, int nsteps,
double tend);
void flux(const int W, double Fx[][W+1], double Fy[][W+1], double T[]
[W+2], double p[][W+1], double F0[]);
void pde(const int W, double E[][W+1], double Fx[][W+1], double Fy[]
[W+1]);
void eos(const int W, double E[][W+1], double T[][W+2], double p[]
[W+1], double Fx[][W+1], double Fy[][W+1]);
```

```
//============================== EQUATION OF STATE
#include "global.h"
#include <stdio.h>

double conduct(double p);  // declare function

void eos(const int W, double E[][W+1], double T[][W+2], double p[]
[W+1], double Fx[][W+1], double Fy[][W+1]){
        // Nodes
        for(int i = 1; i <= M; i++){
                for(int j = 1; j <= M; j++){
                        if(E[i][j] < 0){
                                //if (i==i){printf("im solid\n");}
                                T[i][j] = Tm + E[i][j]/(rho*Cs);
                                p[i][j] = 0;
                        }else if(E[i][j] >= 0 && E[i][j] <= rho*L){
                                //printf("im mushy\n");
                                T[i][j] = Tm;
                                p[i][j] = E[i][j]/(rho*L);
                        }else if(E[i][j] > rho*L){
                                //printf("im liquid\n");
                                T[i][j] = Tm + (E[i][j] - rho*L)/
(rho*Cl);
                                p[i][j] = 1;
                                //printf("%i frac=%f",i, p[i]);
                        }
                /*if (i ==i){
                        printf("Tn+1=%f en+1=%f\n\n", T[1], E[0]);
                }*/
                }
        }

        /*printf("enafterloop=%f\n", E[0]);
        E[0] = E0;
        if(p[1] == 1){p[0] = 1;}*/
        // Boundaries
        double k, R;
                // LEFT AND RIGHT
        for(int j = 1; j <= M; j++){
                k = conduct(p[1][j]);
                R = dx/(2*k);
                T[0][j] = Fx[0][j]*R + T[1][j];
                        // RIGHT
                k = conduct(p[M][j]);
                R = dx/(2*k);
                //T[M+1] = (R*h*Tinf - T[M])/(R*h - 1);
                T[M+1][j] = T[M][j] - Fx[M][j]*R;
                //T[M+1][j] = T0;
        }
```

```
                    // DOWN AND UP
            for(int i = 1; i <= M; i++){
                    k = conduct(p[i][1]);
                    R = dx/(2*k);
                    T[i][0] = Fy[i][0]*R + T[i][1];
                            // RIGHT
                    k = conduct(p[i][M]);
                    R = dx/(2*k);
                    //T[M+1] = (R*h*Tinf - T[M])/(R*h - 1);
                    T[i][M+1] = T[i][M] - Fy[i][M]*R;
                    //T[i][M+1] = T0;
            }

    // CORNERS
    T[0][0] = (T[0][1] + T[1][0])/2.;
    T[0][M+1] = (T[0][M] + T[1][M+1])/2.;
    T[M+1][M+1] = (T[M][M+1] + T[M+1][M])/2.;
    T[M+1][0] = (T[M][0] + T[M+1][1])/2.;

    }
```

```
//============================================= create fluxes at faces
between CV's
#include "global.h"
#include <math.h>
#include <stdio.h>

double conduct(double p);  // declare function

void flux(const int W, double Fx[][W+1], double Fy[][W+1], double T[]
[W+2], double p[][W+1], double F0[]){
        double k, ki, kim, R;
        // Boundaries
                // Left and right
        for(int j = 1; j <= M; j++){
                k = conduct(p[1][j]);
                R = dx/(2*k);
                //Fx[0][j] = -(T[1][j] - Q0)/R; //F_1/2
        Fx[0][j] = - (T[0][j] - Tinf)/(R + 1/h); // left; convective
2-D
                k = conduct(p[M][j]);
                R = dx/(2*k);
                //F[M] = - (T[M] - Tinf)/(R + 1/h); // convective 1-D
                //Fx[M][j] = - (T[M+1][j] - T[M][j])/R; // insulated
2-D
                Fx[M][j] = (T[M][j] - Tinf)/(R + 1/h); // right;
convective 2-D
        }
                // Bottom and top
        for(int i = 1; i <= M; i++){
                k = conduct(p[i][1]);
                R = dx/(2*k);
                //Fy[i][0] = -(T[i][1] - Q0)/R; //F_1/2
        Fy[i][0] = - (T[i][0] - Tinf)/(R + 1/h); // bottom; convective
2-D
                //==================== TOP
=======================================
                k = conduct(p[i][M]);
                R = dx/(2*k);
                if(!BCType){ // checks boundary condition type
                        Fy[i][M] = (T[i][M] - F0[i])/R; // const temp
BC
                        //printf("TEMPBC, Flux = %f\n", Fy[i][M]);
                }else{
                        Fy[i][M] = - F0[i]/(dx);  // const Flux BC
                        //printf("FLUXBC, Flux = %f\n", Fy[i][M]);
                }
                //Fy[i][M] = (T[i][M] - Tinf)/(R + 1/h); //
convective 2-D
        }
        // LEFT TO RIGHT FLUX
        for(int i = 2; i <= M; i++){ // 2-1/2 < i-1/2 < M-1/2
                for(int j = 1; j <= M; j++){
                        // get k and R
                        ki = conduct(p[i][j]); // ki
                        kim = conduct(p[i-1][j]); // ki-1
                        R = dx/(2*ki) + dx/(2*kim); // Ri-1/2
                        // compute flux
                        Fx[i-1][j] = -(T[i][j] - T[i-1][j])/R; //
F_i-1/2
                }
        }
        // UP TO DOWN FLUX
        for(int i = 1; i <= M; i++){ // 2-1/2 < i-1/2 < M-1/2
                for(int j = 2; j <= M; j++){
                        // get k and R
                        ki = conduct(p[i][j]); // ki
                        kim = conduct(p[i][j-1]); // ki-1
                        R = dx/(2*ki) + dx/(2*kim); // Ri-1/2
                        // compute flux
                        Fy[i][j-1] = -(T[i][j] - T[i][j-1])/R; //
F_i-1/2
                }
        }
}
```

```
//=================================== CREATE INITIAL PROFILE
#include "global.h"
#include <stdio.h>
void init(const int W, double T[W+1][W+2], double E[W+1][W+1], double
p[W+1][W+1]){
        // for nodes and boundaries
        for(int i = 0; i <= M+1; i++){
                for(int j = 0; j<= M+1; j++){
                        // Temperatures
                        T[i][j] = T0; // IC's
                        // Energies and liquid fractions
                        if(i > 0 && j > 0 && i < M+1 && j < M+1){ //
NODES ONLY
                                E[i][j] = rho*Cs*(T[i][j] - Tm);
                                p[i][j] = 0;
                        }
                }
        }
}
```

```
// 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;}
        }
}
```

```
// create 2D mesh array
#include "global.h"
void mesh(double X[], double Y[]){
        X[0] = a; // boundaries
        X[M+1] = b;
        Y[0] = a;
        Y[M+1] = b;
        for(int i = 1; i <= M; i++){
                X[i] = a + ((double)i - 0.5)*dx; // within space
                Y[i] = a + ((double)i - 0.5)*dy; // within space
        }
}
```

```
//======================= print solution profile to text file at
certain times
#include "global.h"
#include <stdio.h>
void output(const int W, double X[], double Y[], double T[][W+2],
double Fx[][W+1], double Fy[][W+1], double E[][W+1], double p[][W+1],
double time, int nsteps, double tend){
//      double flux, energy, phase;
        FILE *OUT;
        FILE *TEMP;
        FILE *PHASE;
        FILE *ENTH; // initialize file var
        if(!nsteps){ // if haven't begun time stepping
                OUT = fopen("outputs/values.o", "w"); // new file
                fprintf(OUT,"#nstep time (ms)  width (cm) depth(cm)
energy (J)\n");
                TEMP = fopen("outputs/temp.o", "w"); // new file
                //fprintf(TEMP,"#Temperatures by timesteps\n");
                PHASE = fopen("outputs/phase.o", "w"); // new file
                //fprintf(PHASE,"#Liquid fraction by timestep\n");
                ENTH = fopen("outputs/enth.o", "w"); // new file
                //fprintf(ENTH,"#Enthalpies by timestep\n");
        }
        else{
                OUT = fopen("outputs/values.o", "a"); // old file
                TEMP = fopen("outputs/temp.o", "w"); // old file
                PHASE = fopen("outputs/phase.o", "w"); // old file
                ENTH = fopen("outputs/enth.o", "w"); // old file
        }
        // hidden time, steps, error
        // position and profile in columns
        //=====================================================
TEMPERATURES
        // inside temperatures only, walls not necessary
        for(int j = 1; j <= M; j++){
                for(int i = 1; i <= M; i++){
                        fprintf(TEMP, "%22.15e ", T[i][j]);
                }
                fprintf(TEMP, "\n");
        }
        fprintf(TEMP, "\n");
        fclose(TEMP);
        //===================================================== PHASES
                for(int j = 1; j <= M; j++){
                        for(int i = 1; i <= M; i++){
                                fprintf(PHASE, "%22.15e ", p[i][j]);
                        }
                fprintf(PHASE, "\n");
        }
        fprintf(PHASE, "\n");
        fclose(PHASE);
        //================================================== ENTHALPIES
        // just for funsies
        for(int j = 1; j <= M; j++){
                for(int i = 1; i <= M; i++){
                        fprintf(ENTH, "%22.15e ", E[i][j]);
                }
                fprintf(ENTH, "\n");
        }
        fprintf(ENTH, "\n");
        fclose(ENTH);
        //==================================================
TIMESTAMPS/ETC
        double xl, xr, yd, width, depth, energy;
        int il = M/2, ir = M/2, id = M;
        for(int l = M/2; l <= M; l++){
                for(int k = 1; k <= M; k++){ // bounds for width of
pool
                        if(p[k-1][l] < 1 && p[k+1][l] == 1){
                                if(k < il){
                                        il = k;}
                        }
                        if(p[k-1][l] == 1 && p[k+1][l] < 1){
                                if(k > ir){
                                        ir = k;}
                        }
        }
        }
        for(int k = 1; k <= M; k++){
                if(p[M/2][k-1] < 1 && p[M/2][k+1] == 1){
                        id = k;}
        }
        xl = a + ((double)il - 0.5)*dx;
        xr = a + ((double)ir - 0.5)*dx;
        yd = a + ((double)id - 0.5)*dx;
        width = 100*(xr - xl); // in cm
        depth = 100*(b - yd); // in cm
        energy = Q0*time; // in Joules
        fprintf(OUT, "%i %6.4f %6.4f %6.4f %6.4f\n", nsteps,
1000*time, width, depth, energy);
        //fprintf(OUT, "# Error up to this time: %.15e\n\n", ERR);
        //
        // COMPLETION BAR
        if(nsteps == 0){
                printf("0%%       20%%       40%%       60%%
80%%       100%%\n");
                printf(" ");
        }
        if( time > nbar*tend/50 ){
                //printf("%3.0f%%\n", 100*time/tend);
```

```
                        printf("=");
                        fflush(stdout);
                        nbar = nbar + 1;
                }
                if(width > 2){
                        maxwidth = 1;
                        int barleft = 50 - nbar;
                        for(int g = 0; g <= barleft; g++){
                        printf("=");}
                }


                /*for(int j = M+1; j >= 0; j--){
                        for(int i = 0; i <= M+1; i++){
                                fprintf(OUT, "%22.15e ", T[i][j]);
                        }
                        fprintf(OUT, "\n");
                }*/


                /*for(int i = 0; i <= M+1; i++){
                        if(i == 0){
                                fprintf(OUT, "%18.15e %18.15e %18.15e\n",
X[i], T[i], F[i]);
                        }
                        if(i > 0 && i <= M){
                                fprintf(OUT, "%18.15e %18.15e %18.15e %18.15e
%.3f\n", X[i], T[i], F[i], E[i], p[i]);
                        }else if(i > M){
                                fprintf(OUT, "%.15e %.15e\n", X[i], T[i]);
                        }
                }*/
                //fprintf(OUT, "\n"); // new line to separate times of
printing
                fclose(OUT);
}
```

```
//========================================== Forward Euler (explicit)
#include "global.h"
void pde(const int W, double E[][W+1], double Fx[][W+1], double Fy[]
[W+1]){
    // Energy is conserved within nodes only..
        for(int i = 1; i <= M; i++){
                for(int j = 1; j <= M; j++){
                        E[i][j] = E[i][j] + (dt/dx)*(Fx[i-1][j] -
Fx[i][j]) + (dt/dy)*(Fy[i][j-1] - Fy[i][j]);
                        // Ei + dt/dx(sum fluxes)
                }
        }
}
```

```
//========================== READ INPUTS
#include "global.h"
#include <stdio.h>
void readfile(char filename[], double *factor, double *dtout, double
*tend, int *MM){
        FILE *IN;
    int n = 255;
    char buff[n];
    IN = fopen(filename, "r");

    // TIME
    fgets(buff, n, IN); // section
    fgets(buff, n, IN); // labels
    fscanf(IN, "%lf %lf %lf\n", factor, dtout, tend); // values

    // SPACE
    fgets(buff, n, IN);
    fgets(buff, n, IN);
    fscanf(IN, "%i %lf %lf\n", MM, &a, &b);

    // MATERIAL
    fgets(buff, n, IN);
    fgets(buff, n, IN);
    fscanf(IN, "%lf %lf %lf %lf %lf\n", &rho, &L, &Tm, &h, &Tinf);
    fgets(buff, n, IN);
    fscanf(IN, "%lf %lf %lf %lf\n", &Cs, &Cl, &ks, &kl);

    // IBCs
    fgets(buff, n, IN);
    fgets(buff, n, IN);
    fscanf(IN, "%lf %lf %i %i\n", &T0, &Q0, &BCType, &FType);

    // printf("%i, %f, %f, %f, %f\n", MM, Cs, b, Q0, T0);
    fclose(IN);
}
```