```
//MAE 5150: Coding Project 2
//Max Le
//HEADER FILE WITH FUNCTIONS
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <ctime>
using namespace std;

//DECLARE STRUCTURE
struct Elliptic
{
    double dx ;
    double dy;
    double imax;
    double jmax;
    double ERROR_MAX;
    vector<vector <double> > psi;
    vector<vector <double> > psi_updated;
    double ERROR;
    int maxiter;
    int iter;
    double w ; //relaxation parameter
    double beta;
    double oneby2BetaSquare;
    double betaSquare;
    double psi1;
    double psi3;
    double OneMinusOmega;
    double OmegaBetaSquare;
    double TimesBetaSquare;
    double Omegaby2BetaSquare;
    float timeElapsed;
};

//FUNCTION TO RESET PSI EVERYTIME IT'S CALLED (BC, ZEROS)
void InitializePsi(struct Elliptic *laplace, float wi){
    laplace->dx = 0.2;
    laplace->dy = 0.2;
    laplace->imax = 31;
    laplace->jmax = 21;
    laplace->ERROR_MAX = 0.01; //max allowable error
    laplace->ERROR=0.0;
    laplace->maxiter = 500;
    laplace->iter = 0;
    laplace->psi.resize(laplace->jmax+1);
    laplace->psi_updated.resize(laplace->jmax+1);
    laplace->w = wi;
    laplace->beta = (laplace->dx)/(laplace->dy);
```

```
    laplace->oneby2BetaSquare = (1./(2*(1+pow(laplace->beta,2.))));
    laplace->betaSquare = pow(laplace->beta,2.);
    laplace->psi1 = 0.0;
    laplace->psi3 = 100.00;
    laplace->OneMinusOmega = (1-laplace->w);
    laplace->OmegaBetaSquare = laplace->w*laplace->betaSquare;
    laplace->TimesBetaSquare = 2.*(1+pow(laplace->beta,2.));
    laplace->Omegaby2BetaSquare = (laplace->w/(2.*(1+pow(laplace->beta,2.))));

    for (int j = 1; j<=laplace->jmax;j++){
        laplace->psi[j].resize(laplace->imax+1);
        laplace->psi_updated[j].resize(laplace->imax+1);
    }

    //SET zero to everywhere
    for (int j = 1; j<=laplace->jmax;j++){
        for (int i = 1; i<=laplace->imax;i++){
            laplace->psi[j][i] = 0.0;
            laplace->psi_updated[j][i] = 0.0;       }
    }
    // //bottom BC,initially ZERO everywhere
    for (int i = 7; i<=laplace->imax;i++){
        laplace->psi[1][i] = laplace->psi3;
        laplace->psi_updated[1][i] = laplace->psi3;
    }
}

//THOMAS ALGORITHM
void thomasTriDiagonalX(int j, double a[],double b[],double c[], double d[],
struct Elliptic *laplace){

    int imax = laplace->imax-1;
    double dprime[imax+1];
    double cprime[imax+1];
    dprime[1] = d[1];
    cprime[1] = c[1];

    //FORWARD LOOP
    for (int i = 2; i<=imax;i++){
        dprime[i] = d[i] - ((b[i]*a[i-1])/(dprime[i-1]));
        cprime[i] = c[i] - ((cprime[i-1]*b[i])/(dprime[i-1]));
    }

    laplace->psi_updated[j][imax] = cprime[imax]/dprime[imax];

    //BACKWARD LOOP
    for (int i = imax-1;i>=2;i--){
        laplace->psi_updated[j][i] = (cprime[i]-(a[i]*laplace->psi_updated[j][i
+1]))/(dprime[i]);
    }
}
```

```
//POINT GAUSS SEIDEL
void PointGaussSeidel(struct Elliptic *laplace){
    do{
        laplace->ERROR = 0.0;
        for (int j = 2; j<=laplace->jmax-1;j++){
            for (int i = 2; i<=laplace->imax-1;i++){
                    //Finite Difference, using psi_updated as Latest Data

                    laplace->psi_updated[j][i] = (laplace->oneby2BetaSquare)*
(laplace->psi[j][i+1]+laplace->psi_updated[j][i-1]+(laplace->betaSquare)*
(laplace->psi[j+1][i]+laplace->psi_updated[j-1][i]));

                    //Calculate error, keep doing this until satisfy ERROR MAX
                    laplace->ERROR += abs((laplace->psi_updated[j][i] -
laplace->psi[j][i]));
            }
        }
        //Updating Pupdated with P
        for (int j = 2; j<=laplace->jmax-1;j++){
            for (int i =2; i<=laplace->imax-1;i++){
                laplace->psi[j][i] = laplace->psi_updated[j][i];
            }
        }
        // Make sure BC satisfied, dpsi/dx = 0
        for (int j = 2; j<=laplace->jmax-1;j++){
                laplace->psi[j][laplace->imax] = laplace->psi[j][laplace-
>imax-1];
        }
        //Update the iteration counter
        laplace->iter = laplace->iter + 1;
        }while(laplace->ERROR > laplace->ERROR_MAX);

        printf("Point GS Converged! Max iter is: %d \n", laplace->iter);

        // //PRINTING TO FILE

        FILE * outfile1;
        outfile1 = fopen("ResultsPointGS.dat","w");

        for (int j = 1; j<=laplace->jmax;j++){
            for (int i =1; i<=laplace->imax;i++){
                fprintf(outfile1,"%6.9f\t", laplace->psi[j][i]);
            }
            fprintf(outfile1,"\n");
        }
}

//LINE GAUSS SEIDEL
void LineGaussSeidel(struct Elliptic *laplace){

    //THOMAS PARAMETERS
    //Define vectors for Thomas (in X)
```

```
int imax = laplace->imax;
double ax[imax+1]; //above
double bx[imax+1]; //below
double cx[imax+1];//rhs
double diagonalX[imax+1];//diagonal
//Fill out zero values to all ax.bx.cx.Dx
for (int i = 0;i<=imax+1;i++){
    ax[i] = 0.0;
    bx[i] = 0.0;
    cx[i] = 0.0;
    diagonalX[i] = 0.0;
}
//Fill out values
//For ay,by,Dy from 1 to JMAX for now
//Rewrite value below
for (int i=1; i<=laplace->imax;i++){
    ax[i] = 1.;
    bx[i] = 1.;
    diagonalX[i] = -1.0*laplace->TimesBetaSquare;
}
//Special values rewrite
//Don't use zero elements; zero out these
//Really dont need, because loop at 2 to imax-1
ax[0] = 0.0;
bx[0] = 0.0;
cx[0] = 0.0;
diagonalX[0] = 0.0;
//All the 1st elements = 0
ax[1] = 0.0;
bx[1] = 0.0;
cx[1] = laplace->psi1; //which is 0
diagonalX[1] = 1.0;
//All last values = 0
ax[imax] = 0.0;
bx[imax] = 0.0;
cx[imax] = 0.0;
diagonalX[imax] = 1.0;
//Diagonals, d has full, a misses last, b misses first
ax[imax-1] = 0.0;// a misses last
bx[2] = 0.0; //b misses first

//LINE GAUSS SEIDEL LOOP
do{
    laplace->ERROR = 0.0;
    // Loop through jTH row
    for (int j = 2; j<=laplace->jmax-1;j++){
        //Loop through iTH row
        for (int i = 2; i<=laplace->imax-1;i++){
            // //BOUNDARY CONDITIONS
            //at I = 2, zero
            if (i == 2){
```

```
                    cx[i] = (
                        (-laplace->betaSquare*(laplace->psi[j+1][i]))+
                        (-laplace->betaSquare*laplace->psi_updated[j-1][i])-
                        (laplace->psi1)
                    );
                }
                //at I = IMAX-1, will have psi[IMAX], which needs dPsi/dx = 0
                else if (i == laplace->imax-1){
                    cx[i] = (
                        (-laplace->betaSquare*(laplace->psi[j+1][i]))+
                        (-laplace->betaSquare*laplace->psi_updated[j-1][i])-
                        (laplace->psi_updated[j][i+1])
                    );
                }
                else{
                    cx[i] = (
                        -(laplace->betaSquare*laplace->psi[j+1][i])-
                        (laplace->betaSquare*laplace->psi_updated[j-1]
[i]));
                }
            }
            thomasTriDiagonalX(j,ax,bx,cx,diagonalX,laplace);

            laplace->psi_updated[j][laplace->imax] =
                laplace->psi_updated[j][laplace->imax-1];

            for(int i=1; i<=laplace->imax; i++) {
                laplace->ERROR +=
                    abs((laplace->psi_updated[j][i] - laplace->psi[j][i]));
            }
        }
        //Updating Pupdated with P
        for (int j = 1; j<=laplace->jmax;j++){
            for (int i =1; i<=laplace->imax;i++){
                laplace->psi[j][i] = laplace->psi_updated[j][i];
            }
        }
        //Update the iteration counter
        laplace->iter = laplace->iter + 1;

    }while(laplace->ERROR > laplace->ERROR_MAX);

    printf("Line GS Converged! Max iter is: %d\n", laplace->iter);

    //PRINTING TO FILE
    FILE * outfile2;
    outfile2 = fopen("ResultsLineGS.dat","w");
    for (int j = 1; j<=laplace->jmax;j++){
        for (int i =1; i<=laplace->imax;i++){
            fprintf(outfile2,"%6.9f\t", laplace->psi[j][i]);
        }
        fprintf(outfile2,"\n");
```

```
    }
}


//LINE SOR
void LineSOR(struct Elliptic *laplace){

    //start clock
    clock_t t;

    t = clock();

    int imax = laplace->imax;
    double ax[imax+1]; //above
    double bx[imax+1]; //below
    double cx[imax+1];//rhs
    double diagonalX[imax+1];//diagonal

    //Fill out zero values to all ax.bx.cx.Dx

    for (int i = 0;i<=imax+1;i++){
        ax[i] = 0.0;
        bx[i] = 0.0;
        cx[i] = 0.0;
        diagonalX[i] = 0.0;
    }

    //Fill out values
    for (int i=1; i<=laplace->imax;i++){
        ax[i] = laplace->w;
        bx[i] = laplace->w;
        diagonalX[i] = -1.0*laplace->TimesBetaSquare;
    }

    //Special values rewrite
    //Don't use zero elements; zero out these
    //Really dont need, because loop at 2 to jmax-1
    ax[0] = 0.0;
    bx[0] = 0.0;
    cx[0] = 0.0;
    diagonalX[0] = 0.0;
    //All the 1st elements = 0
    ax[1] = 0.0;
    bx[1] = 0.0;
    cx[1] = laplace->psi1; //which is 0
    diagonalX[1] = 1.0;
    //All last values = 0
    ax[imax] = 0.0;
    bx[imax] = 0.0;
    cx[imax] = 0.0;
    diagonalX[imax] = 1.0;
    //Diagonals, d has full, a misses last, b misses first
```

```
    ax[imax-1] = 0.0;// a misses last
    bx[2] = 0.0; //b misses first

    //LINE GAUSS SEIDEL LOOP
    do{
        laplace->ERROR = 0.0;
        // Loop through iTH row
        for (int j = 2; j<=laplace->jmax-1;j++){
            //Loop through jTH row
            for (int i = 2; i<=laplace->imax-1;i++){
                //BOUNDARY CONDITIONS
                //at I = 2, zero
                if (i == 2){
                    cx[i] = -(laplace->OneMinusOmega*laplace-
>TimesBetaSquare*laplace->psi[j][i])
                        -((laplace->OmegaBetaSquare)*(((laplace->psi[j+1][i]))+
                        (laplace->psi_updated[j-1][i])))-
                        ((laplace->w)*(laplace->psi1))
                    ;
                }

                //at J = JMAX -1, will have psi[JMAX], which needs dPsi/dx = 0
                else if (i == laplace->imax-1){
                    cx[i] = -(laplace->OneMinusOmega*laplace-
>TimesBetaSquare*laplace->psi[j][i])
                        -((laplace->OmegaBetaSquare)*(((laplace->psi[j+1][i]))+
                        (laplace->psi_updated[j-1][i])))-
                        ((laplace->w)*(laplace->psi_updated[j][i+1]));
                }

                else{
                    cx[i] =
                        -(laplace->OneMinusOmega*laplace-
>TimesBetaSquare*laplace->psi[j][i])
                        -((laplace->OmegaBetaSquare)*(((laplace->psi[j+1][i]))+
                        (laplace->psi_updated[j-1][i])));
                }
            }
            thomasTriDiagonalX(j,ax,bx,cx,diagonalX,laplace);

            //BC for dPsi/dx = 0;
            laplace->psi_updated[j][laplace->imax] =
                laplace->psi_updated[j][laplace->imax-1];

            for(int i=1; i<=laplace->imax; i++) {
                laplace->ERROR +=
                    abs((laplace->psi_updated[j][i] - laplace->psi[j][i]));
            }
        }

        //Updating Pupdated with P
```

```
        for (int j = 1; j<=laplace->jmax;j++){
            for (int i =1; i<=laplace->imax;i++){
                laplace->psi[j][i] = laplace->psi_updated[j][i];
            }
        }

        //Update the iteration counter
        laplace->iter = laplace->iter + 1;

    }while(laplace->ERROR > laplace->ERROR_MAX);

    t = clock()-t;

    laplace->timeElapsed = (float)t/(CLOCKS_PER_SEC);

    //End clock
    printf("Line SOR Converged! Max iter is: %d, w = %f at %f seconds \n",
laplace->iter,laplace->w, laplace->timeElapsed);
    FILE *outfile3;
    outfile3 = fopen("ResultsLSOR.dat","w");
    for (int j = 1; j<=laplace->jmax;j++){
        for (int i = 1; i<=laplace->imax;i++){
            fprintf(outfile3,"%6.9f\t", laplace->psi[j][i]);
        }
    fprintf(outfile3,"\n");
    }
}


//POINT SOR
void PointSOR(struct Elliptic *laplace){
    //start clock
    clock_t t;
    t = clock();
    do{
        laplace->ERROR = 0.0;
        for (int j = 2; j<=laplace->jmax-1;j++){
            for (int i = 2; i<=laplace->imax-1;i++){
                    //Finite Difference, using psi_updated as Latest Data

                    laplace->psi_updated[j][i] = ((1-laplace->w)*(laplace->psi
[j][i]))+((laplace->Omegaby2BetaSquare)*(laplace->psi[j][i+1]+laplace-
>psi_updated[j][i-1]+(laplace->betaSquare)*(laplace->psi[j+1][i]+laplace-
>psi_updated[j-1][i])));

                     //Calculate error, keep doing this until satisfy ERROR MAX
                    laplace->ERROR += abs((laplace->psi_updated[j][i] -
laplace->psi[j][i]));
            }
        }

        //Updating Pupdated with P
```

```
        for (int j = 2; j<=laplace->jmax-1;j++){
            for (int i =2; i<=laplace->imax-1;i++){
                laplace->psi[j][i] = laplace->psi_updated[j][i];
            }
        }
        // Make sure BC satisfied, dpsi/dx = 0
        for (int j = 2; j<=laplace->jmax-1;j++){
                laplace->psi[j][laplace->imax] = laplace->psi[j][laplace-
>imax-1];
        }

        //Update the iteration counter
        laplace->iter = laplace->iter + 1;

        }while(laplace->ERROR >= laplace->ERROR_MAX);

        t = clock()-t;

        laplace->timeElapsed = (float)t/(CLOCKS_PER_SEC);

        printf("POINT SOR Converged! Max iter is: %d, w = %f at %f seconds
\n", laplace->iter,laplace->w, laplace->timeElapsed);

        FILE *outfile4;

        outfile4 = fopen("ResultsPSOR.dat","w");
        for (int j = 1; j<=laplace->jmax;j++){
            for (int i = 1; i<=laplace->imax;i++){
                fprintf(outfile4,"%6.9f\t", laplace->psi[j][i]);
            }
            fprintf(outfile4,"\n");
        }


}


void PrintTablesAndCompare(){
    // TABLE FOR BLOWING UP
    //FOR POINT SOR
    {   int nw = 20;
        double dw = 0.1;
        double w0 = 0.1;
        float wi;
        FILE *testPSOR;
        testPSOR = fopen("testPSOR.dat","w");
        FILE *neatTablePSOR;
        neatTablePSOR = fopen("neatTablePSOR.txt","w");
        fprintf(neatTablePSOR,"  w     |  Iteration  | Time[sec] |\n");
        fprintf(neatTablePSOR,"-----------------------------------\n");
        for(int i=0; i<nw; i++){
            Elliptic PSOR;
```

```
            wi = w0 + dw*( (float) i);
            InitializePsi(&PSOR,wi);
            PointSOR(&PSOR);
            fprintf(neatTablePSOR, "%.3f\t|\t%4.0d\t  |\t%f  |
\n",PSOR.w,PSOR.iter,PSOR.timeElapsed);
            fprintf(testPSOR, "%.3f\t%4.0d\t%f
\n",PSOR.w,PSOR.iter,PSOR.timeElapsed);
        }
        fprintf(neatTablePSOR,"\nTable 1- Different values of relaxation
parameter (w) for Point SOR\n\n");
        fclose(testPSOR);
        fclose(neatTablePSOR);
    }
    printf("\n");
    //FOR LINE SOR
    {   int nw = 20;
        double dw = 0.1;
        double w0 = 0.1;
        float wi;
        FILE *testLSOR;
        testLSOR = fopen("testLSOR.dat","w");
        FILE *neatTableLSOR;
        neatTableLSOR = fopen("neatTableLSOR.txt","w");
        fprintf(neatTableLSOR,"  w     |  Iteration  | Time[sec] |\n");
        fprintf(neatTableLSOR,"-----------------------------------\n");
        for(int i=0; i<nw; i++) {
            Elliptic LSOR;
            wi = w0 + dw*( (float) i);
            InitializePsi(&LSOR,wi);
            LineSOR(&LSOR);
            fprintf(neatTableLSOR, "%.3f\t|\t%4.0d\t  |\t%f  |
\n",LSOR.w,LSOR.iter,LSOR.timeElapsed);
            fprintf(testLSOR, "%.3f\t%4.0d\t%f
\n",LSOR.w,LSOR.iter,LSOR.timeElapsed);
        }
        fprintf(neatTableLSOR,"\nTable 2- Different values of relaxation
parameter (w) for Line SOR\n\n");
        fclose(testLSOR);
        fclose(neatTableLSOR);
    }
}

//END HEADER FILE WITH FUNCTIONS
```