```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#include <math.h>

#include "includes/libpoisson.h"

//#define N 258

int main(int argc, char *argv[])
//int main(void)
{
    int N;
    if(argc != 2)
    {
        errorMsg("Erro na execucao!\nEx. ./[nome_executavel] [tamanho_da_malha+2]\n");
        return (1);
    }
    else
    {
        N = atoi(argv[1]);
    }

    /*
     * Fluxos atuais e antigos em cada um dos lados da célula espacial
     * up (Uper), dn (Down), lf (Left), rh (Right)
     */
    nodeSides *q, *q_old;
    //nodeSides **mq,  **mq_old;

    q = criaVetorNode(N, N);
    //mq = criaMatrizNode(N, N, q);
    q_old = criaVetorNode(N, N);
    //mq_old = criaMatrizNode(N, N, q_old);

    //mq = montaMatrizNode(N, N);
    //mq_old = montaMatrizNode(N, N);

    /*
     *  Multiplicadores de Lagrange em cada um dos lados da célula espacial
     */
    nodeSides *l, *l_old;
    //nodeSides **ml, **ml_old;

    l = criaVetorNode(N, N);
    //ml = criaMatrizNode(N, N, l);
    l_old = criaVetorNode(N, N);
    //ml_old = criaMatrizNode(N, N, l_old);

    //ml = montaMatrizNode(N, N);
    //ml_old = montaMatrizNode(N, N);

    /*
     *  Betas da condição de Robin em cada um dos lados da célula espacial
     */

    nodeSides *beta;
    //nodeSides **mbeta;

    beta = criaVetorNode(N, N);
    //mbeta = criaMatrizNode(N, N, beta);

    //mbeta = montaMatrizNode(N, N);

    /*
     *  Pressões atuais e antigas em cada uma das células
     */

    double *p, *p_old;
```

```c
    //double **mp, **mp_old;

    p = criaVetor(N, N);
    //mp = criaMatriz(N, N, p);
    p_old = criaVetor(N, N);
    //mp_old = criaMatriz(N, N, p_old);

    //mp = montaMatriz(N, N);
    //mp_old = montaMatriz(N, N);

    /*
     *  Parametros materiais da grade
     */

    nodeMaterial *pMat;
    //nodeMaterial **mpMat;

    pMat = criaVetorMaterial(N, N);
    //mpMat = criaMatrizMaterial(N, N, pMat);

    //mpMat = montaMatrizMaterial(N, N);

    double lsize = 25600.00; /* dimensão da regiao */

    /* Variáveis para auxiliar na contagem do tempo*/
    double start, stop;
    double startTime, elapsedTime;
    double clockZero = 0.0;
    clock_t ticks1, ticks2;
    /* Variáveis auxiliares na implementação */
    int i, j, k, n;
    double  h, aux,
            c = 1.,                 /* Valor para o calculo de beta */
            Media, erro,            /* Media das pressoes e erro na norma */
            Keff;                   /* Valor medio da permeabilidade */
    //double **mp_aux;                /* Ponteiros para troca */
    double *p_aux;
    //nodeSides **mq_aux;
    nodeSides *q_aux;

    start = omp_get_wtime();
    startTime = walltime( &clockZero );
    ticks1 = clock();

    /*
    * Inicialização das variaveis
    */



    for(i=1; i < N-1; i++)
        for(j=1; j < N-1; j++)
        {
            pMat[i*N+j].f = 0.0;
            //mpMat[i][j].f = 0.0;
            l[i*N+j].dn = l[i*N+j].lf = l[i*N+j].rh = l[i*N+j].up = 0.0;
            //ml[i][j].dn = ml[i][j].lf = ml[i][j].rh = ml[i][j].up = 0.0;
            q[i*N+j].dn = q[i*N+j].lf = q[i*N+j].rh = q[i*N+j].up = 0.0;
            //mq[i][j].dn = mq[i][j].lf = mq[i][j].rh = mq[i][j].up = 0.0;
            p[i*N+j]=0.0;
            //mp[i][j]=0.0;
        }

    /*
     * Cálculo de algumas variáveis auxiliares para o valor de permeabilidade
     * en blocos verticais
     */
    n = N-2;
    h = lsize/n;
    k = n/2;
```

```c

    for(i=1; i<=k; i++)
        for(j=1; j<N-1; j++)
        {
            pMat[i*N+j].perm = 1.0e-10;
            //mpMat[i][j].perm = 1.0e-10;
            pMat[(i+k)*N+j].perm = 1.0e-11;
            //mpMat[i+k][j].perm = 1.0e-11;
        }

    /*
     * Calcula os Beta da Condicao de Robin
     */
    k = N+1; //[1][1]
    Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
    //Keff = (2*mpMat[1][1].perm*mpMat[1][2].perm)/(mpMat[1][1].perm + mpMat[1][2].perm);
    beta[k].up = c*h/Keff;
    //mbeta[1][1].up = c*h/Keff;
    Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
    //Keff = (2*mpMat[1][1].perm*mpMat[2][1].perm)/(mpMat[1][1].perm + mpMat[2][1].perm);
    beta[k].rh = c*h/Keff;
    //mbeta[1][1].rh = c*h/Keff;

    k = N+n; //[1][n]
    Keff = (2*pMat[k].perm*pMat[k-1].perm)/(pMat[k].perm + pMat[k-1].perm);
    //Keff = (2*mpMat[1][n].perm*mpMat[1][n-1].perm)/(mpMat[1][n].perm + mpMat[1][n-1].perm);
    beta[k].dn = c*h/Keff;
    //mbeta[1][n].dn = c*h/Keff;
    Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
    //Keff = (2*mpMat[1][n].perm*mpMat[2][n].perm)/(mpMat[1][n].perm + mpMat[2][n].perm);
    beta[k].rh = c*h/Keff;
    //mbeta[1][n].rh = c*h/Keff;

    k = n*N+1; //[n][1]
    Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
    //Keff = (2*mpMat[n][1].perm*mpMat[n][2].perm)/(mpMat[n][1].perm + mpMat[n][2].perm);
    beta[k].up = c*h/Keff;
    //mbeta[n][1].up = c*h/Keff;
    Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
    //Keff = (2*mpMat[n][1].perm*mpMat[n-1][1].perm)/(mpMat[n][1].perm + mpMat[n-1][1].perm);
    beta[k].lf = c*h/Keff;
    //mbeta[n][1].lf = c*h/Keff;

    k = n*N+n; //[n][n]
    Keff = (2*pMat[k].perm*pMat[k-1].perm)/(pMat[k].perm + pMat[k-1].perm);
    //Keff = (2*mpMat[n][n].perm*mpMat[n][n-1].perm)/(mpMat[n][n].perm + mpMat[n][n-1].perm);
    beta[k].dn = c*h/Keff;
    //mbeta[n][n].dn = c*h/Keff;
    Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
    //Keff = (2*mpMat[n][n].perm*mpMat[n-1][n].perm)/(mpMat[n][n].perm + mpMat[n-1][n].perm);
    beta[k].lf = c*h/Keff;
    //mbeta[n][n].lf = c*h/Keff;

    for (i=2; i<n; i++)
    {
        k = i*N+1; //[i][1]
        Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
        //Keff = (2*mpMat[i][1].perm*mpMat[i][2].perm)/(mpMat[i][1].perm + mpMat[i][2].perm);
        beta[k].up= c*h/Keff;
        //mbeta[i][1].up= c*h/Keff;
        Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
        //Keff = (2*mpMat[i][1].perm*mpMat[i-1][1].perm)/(mpMat[i][1].perm + mpMat[i-1][1].perm);
        beta[k].lf= c*h/Keff;
        //mbeta[i][1].lf= c*h/Keff;
```

```c
        Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
        //Keff = (2*mpMat[i][1].perm*mpMat[i+1][1].perm)/(mpMat[i][1].perm + mpMat[i+1][1].perm);
        beta[k].rh= c*h/Keff;
        //mbeta[i][1].rh= c*h/Keff;

        k = i*N+n; //[i][n]
        Keff = (2*pMat[k].perm*pMat[k-1].perm)/(pMat[k].perm + pMat[k-1].perm);
        //Keff = (2*mpMat[i][n].perm*mpMat[i][n-1].perm)/(mpMat[i][n].perm + mpMat[i][n-1].perm);
        beta[k].dn= c*h/Keff;
        //mbeta[i][n].dn= c*h/Keff;
        Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
        //Keff = (2*mpMat[i][n].perm*mpMat[i-1][n].perm)/(mpMat[i][n].perm + mpMat[i-1][n].perm);
        beta[k].lf= c*h/Keff;
        //mbeta[i][n].lf= c*h/Keff;
        Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
        //Keff = (2*mpMat[i][n].perm*mpMat[i+1][n].perm)/(mpMat[i][n].perm + mpMat[i+1][n].perm);
        beta[k].rh= c*h/Keff;
        //mbeta[i][n].rh= c*h/Keff;

        k = N+i; //[1][i]
        Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
        //Keff = (2*mpMat[1][i].perm*mpMat[1][i+1].perm)/(mpMat[1][i].perm + mpMat[1][i+1].perm);
        beta[k].up= c*h/Keff;
        //mbeta[1][i].up= c*h/Keff;
        Keff = (2*pMat[k].perm*pMat[k-1].perm)/(pMat[k].perm + pMat[k-1].perm);
        //Keff = (2*mpMat[1][i].perm*mpMat[1][i-1].perm)/(mpMat[1][i].perm + mpMat[1][i-1].perm);
        beta[k].dn= c*h/Keff;
        //mbeta[1][i].dn= c*h/Keff;
        Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
        //Keff = (2*mpMat[1][i].perm*mpMat[2][i].perm)/(mpMat[1][i].perm + mpMat[2][i].perm);
        beta[k].rh= c*h/Keff;
        //mbeta[1][i].rh= c*h/Keff;

        k = n*N+i; //[n][i]
        Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
        //Keff = (2*mpMat[n][i].perm*mpMat[n][i+1].perm)/(mpMat[n][i].perm + mpMat[n][i+1].perm);
        beta[k].up= c*h/Keff;
        //mbeta[n][i].up= c*h/Keff;
        Keff = (2*pMat[k].perm*pMat[k-1].perm)/(pMat[k].perm + pMat[k-1].perm);
        //Keff = (2*mpMat[n][i].perm*mpMat[n][i-1].perm)/(mpMat[n][i].perm + mpMat[n][i-1].perm);
        beta[k].dn= c*h/Keff;
        //mbeta[n][i].dn= c*h/Keff;
        Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
        //Keff = (2*mpMat[n][i].perm*mpMat[n-1][i].perm)/(mpMat[n][i].perm + mpMat[n-1][i].perm);
        beta[k].lf= c*h/Keff;
        //mbeta[n][i].lf= c*h/Keff;

        for(j=2; j<n; j++)
        {
            k = i*N+j; //[i][j]
            Keff = (2*pMat[k].perm*pMat[k+1].perm)/(pMat[k].perm + pMat[k+1].perm);
            //Keff = (2*mpMat[i][j].perm*mpMat[i][j+1].perm)/(mpMat[i][j].perm + mpMat[i][j+1].perm);
            beta[k].up= c*h/Keff;
            //mbeta[i][j].up= c*h/Keff;
            Keff = (2*pMat[k].perm*pMat[k-1].perm)/(pMat[k].perm + pMat[k-1].perm);
            //Keff = (2*mpMat[i][j].perm*mpMat[i][j-1].perm)/(mpMat[i][j].perm + mpMat[i][j-1].perm);
            beta[k].dn= c*h/Keff;
            //mbeta[i][j].dn= c*h/Keff;
```

```
            Keff = (2*pMat[k].perm*pMat[k+N].perm)/(pMat[k].perm + pMat[k+N].perm);
            //Keff = (2*mpMat[i][j].perm*mpMat[i+1][j].perm)/(mpMat[i][j].perm + mpMat[i+1][j].perm);
            beta[k].rh= c*h/Keff;
            //mbeta[i][j].rh= c*h/Keff;
            Keff = (2*pMat[k].perm*pMat[k-N].perm)/(pMat[k].perm + pMat[k-N].perm);
            //Keff = (2*mpMat[i][j].perm*mpMat[i-1][j].perm)/(mpMat[i][j].perm + mpMat[i-1][j].perm);
            beta[k].lf= c*h/Keff;
            //mbeta[i][j].lf= c*h/Keff;
        }
    }

    /*
     * Inicializando valores da fonte
     */
    pMat[N+1].f=1.0e-7;
    //mpMat[1][1].f=1.0e-7;
    pMat[n*N+n].f=-1.0e-7;
    //mpMat[n][n].f=-1.0e-7;

    /*
     * calculo de parâmetros que nao dependem das iterações
     */

    aux = 1/h;

    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
        {
            pMat[i*N+j].shi=2*pMat[i*N+j].perm*aux;
            //mpMat[i][j].shi=2*mpMat[i][j].perm*aux;
            pMat[i*N+j].f*=h;
            //mpMat[i][j].f*=h;
        }

    /*
     * Ciclo até convergência do problema
     */
    k = 0;   // quantidade de iterações

    do
    {
        /*
        mp_aux = mp;
        mp = mp_old;
        mp_old = mp_aux;

        mq_aux = mq;
        mq = mq_old;
        mq_old = mq_aux;

        mq_aux = ml;
        ml = ml_old;
        ml_old = mq_aux;
        */

        p_aux = p;
        p = p_old;
        p_old = p_aux;

        q_aux = q;
        q = q_old;
        q_old = q_aux;

        q_aux = l;
        l = l_old;
        l_old = q_aux;

        k++;
```

```
        //printf("Iteração %d \n",k);

        /*Cálculo da pressão e dos fluxos em cada elemento */

        /*Canto inferior esquerdo [1][1]*/
        //canto_d_l(1, 1, mpMat, mbeta, mq, mq_old, ml_old, mp);
        canto_d_lArray(1, 1, N, pMat, beta, q, q_old, l_old, p);

        /*Canto superior esquerdo [1][N-2]*/
        //canto_u_l(1, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
        canto_u_lArray(1, n, N, pMat, beta, q, q_old, l_old, p);

        /*Canto inferior direito [N-2][1]*/
        //canto_d_r(n, 1, mpMat, mbeta, mq, mq_old, ml_old, mp);
        canto_d_rArray(n, 1, N, pMat, beta, q, q_old, l_old, p);

        /*Canto superior direito [N-2][N-2]*/
        //canto_u_r(n, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
        canto_u_rArray(n, n, N, pMat, beta, q, q_old, l_old, p);

        /*Fronteira U [2...N-3][N-2]*/
        //fronteira_u(n, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
        fronteira_uArray(N, n, pMat, beta, q, q_old, l_old, p);

        /*Fronteira D [2...N-3][1]*/
        //fronteira_d(n, 1, mpMat, mbeta, mq, mq_old, ml_old, mp);
        fronteira_dArray(N, 1, pMat, beta, q, q_old, l_old, p);

        /*Fronteira R [N-2][2...N-3]*/
        //fronteira_r(n, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
        fronteira_rArray(n, N, pMat, beta, q, q_old, l_old, p);

        /*Fronteira L [1][2...N-3]*/
        //fronteira_l(1, n, mpMat, mbeta, mq, mq_old, ml_old, mp);
        fronteira_lArray(1, N, pMat, beta, q, q_old, l_old, p);

        /*Elementos internos [2..N-3][2..N-3]*/
        //internos(n, mpMat, mbeta, mq, mq_old, ml_old, mp);
        internosArray(N, pMat, beta, q, q_old, l_old, p);

        /* Atualização dos multiplicadores de lagrange e calculando a média da pressão*/
        //Media = lagrangeUpdate(n, mbeta, mq, mq_old, ml, ml_old, mp);
        Media = lagrangeUpdateArray(N, beta, q, q_old, l, l_old, p);

        /* Impondo a média zero na distriubição de pressões
         * e cálculo de verificação de convergência
         */
        //erro = mediaZero(n, Media, ml, mp, mp_old);
        erro = mediaZeroArray(N, Media, l, p, p_old);

    }
    while(erro > 1e-5);


    //free(mpMat);
    free(pMat);
    //free(mp_old);
    free(p_old);
    //free(mp);
    free(p);
    //free(mbeta);
    free(beta);
    //free(ml_old);
    free(l_old);
    //free(ml);
    free(l);
    //free(mq_old);
    free(q_old);
    //free(mq);
    free(q);
```

```
    /*
    for(i=1; i < N-1; i++){
        free(mpMat[i]);
        free(mp_old[i]);
        free(mp[i]);
        free(mbeta[i]);
        free(ml_old[i]);
        free(ml[i]);
        free(mq_old[i]);
        free(mq[i]);
    }
    free(mpMat);
    free(mp_old);
    free(mp);
    free(mbeta);
    free(ml_old);
    free(ml);
    free(mq_old);
    free(mq);
    */

    ticks2 = clock();
    elapsedTime = walltime( &startTime );
    stop = omp_get_wtime();

    /*Dados finais mostrados na tela*/

    printf("\n------ MALHA %d x %d",n,n);
    printf("\n------ PRESSAO ------- ITERACAO %d ------ ERRO %6.4E", k, erro);
    printf("\n------ CPU TIME %ld, %ld, %g,  %g, %.4g", ticks2, ticks1,
    (ticks1+1.-1.)/CLOCKS_PER_SEC, (ticks2+1.-1.)/CLOCKS_PER_SEC,
    (ticks2+1.-1.)/CLOCKS_PER_SEC - (ticks1+1.-1.)/CLOCKS_PER_SEC);
    printf("\n------ TOTAL TIME  %.6f\n\n",elapsedTime);
    printf("\n------ TOTAL TIME  %.6f\n\n",stop - start);

    return 0; /* fim da funcao main */


    printf("Oi\n");

    return 0;


}
```