```c
#include <stdio.h>
#include <stdlib.h>
//#include<complex.h>
#include <math.h>
#include <fftw3.h>


void transformer(double * in, int Rx, int Ry, double *res_re, double * res_im){
  fftw_complex * spatial_repr;
  fftw_complex * freq_repr;
  fftw_plan plan;

  /*double * res_re;
  double * res_im;*/

  int i,j,x,y;

  spatial_repr = malloc(sizeof(fftw_complex)*Rx*Ry);
  freq_repr = malloc(sizeof(fftw_complex)*Rx*Ry);
  //printf("spatial_repr\n" );
  for(i=0;i<Rx*Ry;i++){
    spatial_repr[i][0] = in[i];
    spatial_repr[i][1] = 0;
    //printf("%f+i*(%f)\n",spatial_repr[i][0],spatial_repr[i][1] );
  }

  plan = fftw_plan_dft_2d(Rx,Ry,spatial_repr,freq_repr,FFTW_FORWARD,FFTW_ESTIMATE);
  fftw_execute(plan);

  /*res_re = malloc(sizeof(double)*Rx*Ry);
  res_im = malloc(sizeof(double)*Rx*Ry);*/

  //printf("freq_repr\n" );
  for(i=0;i<Rx*Ry;i++){
    res_re[i] = freq_repr[i][0];
    res_im[i] = freq_repr[i][1];
    //printf("%f+i*(%f)\n",freq_repr[i][0],freq_repr[i][1] );
  }

  for(j=0;j<Ry;j++){
    for(i=0; i<Rx;i++){
      x=i;
      y=j;
      if(i<Rx/2 && j<Ry/2){
        x=i+Rx/2;
        y=j+Ry/2;
      }
      if(i>=Rx/2 && j<Ry/2){
        x=i-Rx/2;
        y=j+Ry/2;
      }
      if(i<Rx/2 && j>=Ry/2){
        x=i+Rx/2;
        y=j-Ry/2;
      }
      if(i>=Rx/2 && j>=Ry/2){
        x=i+Rx/2;
        y=j+Ry/2;
      }
      printf("y*Rx+x = %d, j*Rx+i = %d\n",y*Rx+x,j*Rx+i );
      res_re[y*Rx+x]=freq_repr[j*Rx+i][0];
      res_im[y*Rx+x]=freq_repr[j*Rx+i][1];
      //printf("%f+i*(%f)\n",res_re[i],res_im[i]);
    }
  }

  fftw_destroy_plan(plan);
  fftw_free(spatial_repr);
  fftw_free(freq_repr);

}
```

```c
double* inverse(double * reIn, double * imIn, int Rx, int Ry){
  fftw_complex * spatial_repr;
  fftw_complex * freq_repr;
  fftw_plan plan;
  double * out;

  int i,j,x,y;

  spatial_repr = malloc(sizeof(fftw_complex)*Rx*Ry);
  freq_repr = malloc(sizeof(fftw_complex)*Rx*Ry);

  for(j=0;j<Ry;j++){
    for(i=0; i<Rx;i++){
      x=i;
      y=j;
      if(i<Rx/2 && j<Ry/2){
        x=i+Rx/2;
        y=j+Ry/2;
      }
      if(i>=Rx/2 && j<Ry/2){
        x=i-Rx/2;
        y=j+Ry/2;
      }
      if(i<Rx/2 && j>=Ry/2){
        x=i+Rx/2;
        y=j-Ry/2;
      }
      if(i>=Rx/2 && j>=Ry/2){
        x=i+Rx/2;
        y=j+Ry/2;
      }
      freq_repr[j*Rx+i][0]=reIn[y*Rx+x];
      freq_repr[j*Rx+i][1]=imIn[y*Rx+x];
    }
  }
  plan = fftw_plan_dft_2d(Rx,Ry,freq_repr,spatial_repr,FFTW_BACKWARD,FFTW_ESTIMATE);
  fftw_execute(plan);

  out = malloc(sizeof(double)*Rx*Ry);
  for(i=0;i<Rx*Ry;i++){
    out[i]=spatial_repr[i][0]/(Rx*Ry);
    //printf("%f\n",out[i] );
  }
  return out;
  fftw_destroy_plan(plan);
  fftw_free(spatial_repr);
  fftw_free(freq_repr);
}

void filtre(double* reImg,double*imImg, double *reFiltre,double * imFiltre, int Rx, int Ry){
  double a, b, c, d;
  int i;

  for(i=0; i<Rx*Ry;i++){
    a = reImg[i];
    b = imImg[i];
    c = reFiltre[i];
    d = imImg[i];

    reImg[i] = a*c-b*d;
    imImg[i] = b*c + a*d;
  }
}

void fitreRamLak(double * reFiltre, double * imFiltre, int Rx, int Ry){
  double * f;
  int i;
  f = malloc(sizeof(double)*Rx*Ry);
  for (i=0;i<Rx*Ry;i++){
    if(i<(Rx*Ry)/2){
```

```
      f[i] = i;
    }
    else{
      f[i] = (Rx*Ry)-i;
    }
    fftw_complex * filtre_av;
    fftw_complex * filtre_ap;
    fftw_plan plan;

    filtre_av = fftw_malloc(sizeof(fftw_complex)*Rx*Ry);
    filtre_ap = fftw_malloc(sizeof(fftw_complex)*Rx*Ry);

    plan = fftw_plan_dft_1d(Rx*Ry,filtre_av,filtre_ap,FFTW_FORWARD,FFTW_ESTIMATE);
    fftw_execute(plan);

    for(i=0;i<Rx*Ry;i++){
      reFiltre[i]=filtre_ap[i][0];
      imFiltre[i]=filtre_ap[i][1];
    }

  }
}

double * miseForme(double ** matrice, int Rx, int Ry){
  double * out;
  int i,j;
  out = malloc(sizeof(double)*Rx*Ry);
  for(j=0;j<Ry;j++){
    for(i=0;i<Rx;i++){
      out[i+j*Rx]=matrice[i][j];
    }
  }
  return out;
}

double ** miseFormeInv(double * matrice, int Rx, int Ry){
  double ** out;
  int i,j;
  out = malloc(sizeof(double)*Ry);

  for(j=0;j<Ry;j++){
    out[j]=malloc(sizeof(double)*Rx);
    for(i=0;i<Rx;i++){
      printf("(%d,%d)\n",j,i);
      out[i][j]=matrice[i+j*Rx];
    }
  }
  return out;
}
```