```c
/*-------------------------------------------------------*/
/* module  : FonctionDemo1.c                             */
/* auteur  : Mignotte Max                                */
/* date    : 18/05/2010                                  */
/* langage : C                                           */
/* labo    : DIRO                                        */
/*-------------------------------------------------------*/

/*-------------------------------------------------*/
/* FICHIERS INCLUS -------------------------------*/
/*-------------------------------------------------*/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "FonctionDemo1.h"

/*-------------------------------------------------*/
/* FONCTIONS -------------------------------------*/
/*-------------------------------------------------*/
/*-----------------------------------------------------------*/
/*  Alloue de la memoire pour une matrice 1d de float      */
/*-----------------------------------------------------------*/
float* fmatrix_allocate_1d(int hsize)
 {
  float* matrix;

  matrix=(float*)malloc(sizeof(float)*hsize);
  if (matrix==NULL) printf("probleme d'allocation memoire");

  return matrix;
 }

/*-----------------------------------------------------------*/
/*  Alloue de la memoire pour une matrice 2d de float       */
/*-----------------------------------------------------------*/
float** fmatrix_allocate_2d(int vsize,int hsize)
 {
  int i;
  float** matrix;
  float *imptr;

  matrix=(float**)malloc(sizeof(float*)*vsize);
  if (matrix==NULL) printf("probleme d'allocation memoire");

  imptr=(float*)malloc(sizeof(float)*hsize*vsize);
  if (imptr==NULL) printf("probleme d'allocation memoire");

  for(i=0;i<vsize;i++,imptr+=hsize) matrix[i]=imptr;
  return matrix;
 }

/*-----------------------------------------------------------*/
/* Libere la memoire de la matrice 1d de float              */
/*-----------------------------------------------------------*/
void free_fmatrix_1d(float* pmat)
 {
  free(pmat);
 }

//-----------------------------------------------------------*/
/* Libere la memoire de la matrice 2d de float              */
/*-----------------------------------------------------------*/
void free_fmatrix_2d(float** pmat)
 {
  free(pmat[0]);
  free(pmat);
```

```
 }

/*-------------------------------------------------------------*/
/* Chargement de l'image de nom <name> (en pgm)                */
/*-------------------------------------------------------------*/
float** LoadImagePgm(char* name,int *length,int *width)
 {
  int i,j,k;
  unsigned char var;
  char buff[NBCHAR];
  float** mat;

  char stringTmp1[NBCHAR],stringTmp2[NBCHAR],stringTmp3[NBCHAR];

  int ta1,ta2,ta3;
  FILE *fic;

  /*-----nom du fichier pgm-----*/
  strcpy(buff,name);
  strcat(buff,".pgm");
  printf("---> Ouverture de %s",buff);

  /*----ouverture du fichier----*/
  fic=fopen(buff,"r");
  if (fic==NULL)
    { printf("\n- Grave erreur a l'ouverture de %s  -\n",buff);
      exit(-1); }

  /*--recuperation de l'entete--*/
  fgets(stringTmp1,100,fic);
  fgets(stringTmp2,100,fic);
  fscanf(fic,"%d %d",&ta1,&ta2);
  fscanf(fic,"%d\n",&ta3);

  /*--affichage de l'entete--*/
  printf("\n\n--Entete--");
  printf("\n----------");
  printf("\n%s%s%d %d \n%d\n",stringTmp1,stringTmp2,ta1,ta2,ta3);

  *length=ta1;
  *width=ta2;
  mat=fmatrix_allocate_2d(*length,*width);

  /*--chargement dans la matrice--*/
     for(i=0;i<*length;i++)
      for(j=0;j<*width;j++)
        { fread(&var,1,1,fic);
          mat[i][j]=var; }

   /*---fermeture du fichier---*/
  fclose(fic);

  return(mat);
 }


/*-------------------------------------------------------------*/
/* Sauvegarde de l'image de nom <name> au format pgm           */
/*-------------------------------------------------------------*/
void SaveImagePgm(char* name,float** mat,int length,int width)
 {
  int i,j,k;
  char buff[NBCHAR];
  FILE* fic;
  time_t tm;

  /*--extension--*/
  strcpy(buff,name);
  strcat(buff,".pgm");
```

```
  /*--ouverture fichier--*/
  fic=fopen(buff,"w");
    if (fic==NULL)
        { printf(" Probleme dans la sauvegarde de %s",buff);
          exit(-1); }
  printf("\n Sauvegarde de %s au format pgm\n",name);

  /*--sauvegarde de l'entete--*/
  fprintf(fic,"P5");
  if ((ctime(&tm))==NULL) fprintf(fic,"\n#\n");
  else fprintf(fic,"\n# IMG Module, %s",ctime(&tm));
  fprintf(fic,"%d %d",width,length);
  fprintf(fic,"\n255\n");

  /*--enregistrement--*/
     for(i=0;i<length;i++)
      for(j=0;j<width;j++)
        fprintf(fic,"%c",(char)mat[i][j]);

  /*--fermeture fichier--*/
   fclose(fic);
 }

/*--------------*/
/* FOURIER -----*/
/*--------------*/
/*-------------------------------------------------*/
/*  FOURN ------------------------------------------*/
/*-------------------------------------------------*/
void fourn(float data[], unsigned long nn[], int ndim, int isign)
{
        int idim;
        unsigned long i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
        unsigned long ibit,k1,k2,n,nprev,nrem,ntot;
        float tempi,tempr;
        double theta,wi,wpi,wpr,wr,wtemp;

        for (ntot=1,idim=1;idim<=ndim;idim++)
                ntot *= nn[idim];
        nprev=1;
        for (idim=ndim;idim>=1;idim--) {
                n=nn[idim];
                nrem=ntot/(n*nprev);
                ip1=nprev << 1;
                ip2=ip1*n;
                ip3=ip2*nrem;
                i2rev=1;
                for (i2=1;i2<=ip2;i2+=ip1) {
                        if (i2 < i2rev) {
                                for (i1=i2;i1<=i2+ip1-2;i1+=2) {
                                        for (i3=i1;i3<=ip3;i3+=ip2) {
                                                i3rev=i2rev+i3-i2;
                                                SWAP(data[i3],data[i3rev]);
                                                SWAP(data[i3+1],data[i3rev+1]);
                                        }
                                }
                        }
                        ibit=ip2 >> 1;
                        while (ibit >= ip1 && i2rev > ibit) {
                                i2rev -= ibit;
                                ibit >>= 1;
                        }
                        i2rev += ibit;
                }
                ifp1=ip1;
                while (ifp1 < ip2) {
                        ifp2=ifp1 << 1;
                        theta=isign*6.28318530717959/(ifp2/ip1);
```

```
                        wtemp=sin(0.5*theta);
                        wpr = -2.0*wtemp*wtemp;
                        wpi=sin(theta);
                        wr=1.0;
                        wi=0.0;
                        for (i3=1;i3<=ifp1;i3+=ip1) {
                                for (i1=i3;i1<=i3+ip1-2;i1+=2) {
                                        for (i2=i1;i2<=ip3;i2+=ifp2) {
                                                k1=i2;
                                                k2=k1+ifp1;
                                                tempr=(float)wr*data[k2]-(float)wi*dat
a[k2+1];
                                                tempi=(float)wr*data[k2+1]+(float)wi*d
ata[k2];
                                                data[k2]=data[k1]-tempr;
                                                data[k2+1]=data[k1+1]-tempi;
                                                data[k1] += tempr;
                                                data[k1+1] += tempi;
                                        }
                                }
                                wr=(wtemp=wr)*wpr-wi*wpi+wr;
                                wi=wi*wpr+wtemp*wpi+wi;
                        }
                        ifp1=ifp2;
                }
                nprev *= n;
        }
}


/*-------------------------------------------------------------*/
/* FFTDD                                                       */
/*-------------------------------------------------------------*/
void FFTDD(float** mtxR,float** mtxI,int lgth, int wdth)
{
 int i,j;
 int posx,posy;

 float* data;
 float* ImgFreqR;
 float* ImgFreqI;
 unsigned long* nn;

 /*allocation memoire*/
 data=(float*)malloc(sizeof(float)*(2*wdth*lgth)+1);
 ImgFreqR=(float*)malloc(sizeof(float)*(wdth*lgth));
 ImgFreqI=(float*)malloc(sizeof(float)*(wdth*lgth));
 nn=(unsigned long*)malloc(sizeof(unsigned long)*(FFT2D+1));

 /*Remplissage de nn*/
 nn[1]=lgth; nn[2]=wdth;

 /*Remplissage de data*/
 for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
   { data[2*(i*lgth+j)+1]=mtxR[i][j];
     data[2*(i*lgth+j)+2]=mtxI[i][j]; }

 /*FFTDD*/
 fourn(data,nn,FFT2D,FFT);

 /*Remplissage*/
 for(i=0;i<(wdth*lgth);i++)
  { ImgFreqR[i]=data[(2*i)+1];
    ImgFreqI[i]=data[(2*i)+2];  }

 /*Conversion en Matrice*/
 for(i=0;i<(wdth*lgth);i++)
  { posy=(int)(i/wdth);
    posx=(int)(i%wdth);
```

```
    mtxR[posy][posx]=ImgFreqR[i];
    mtxI[posy][posx]=ImgFreqI[i];}

 /*Liberation memoire*/
 free(data);
 free(ImgFreqR);
 free(ImgFreqI);
 free(nn);
}


/*-------------------------------------------------------------*/
/* IFFTDD                                                      */
/*-------------------------------------------------------------*/
void IFFTDD(float** mtxR,float**  mtxI,int lgth,int wdth)
{
 int i,j;
 int posx,posy;

 float* data;
 float* ImgFreqR;
 float* ImgFreqI;
 unsigned long* nn;

 /*allocation memoire*/
 data=(float*)malloc(sizeof(float)*(2*wdth*lgth)+1);
 ImgFreqR=(float*)malloc(sizeof(float)*(wdth*lgth));
 ImgFreqI=(float*)malloc(sizeof(float)*(wdth*lgth));
 nn=(unsigned long*)malloc(sizeof(unsigned long)*(FFT2D+1));

 /*Recadrege*/

 /*Remplissage de nn*/
 nn[1]=lgth; nn[2]=wdth;

 /*Remplissage de data*/
 for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
   { data[2*(i*lgth+j)+1]=mtxR[i][j];
     data[2*(i*lgth+j)+2]=mtxI[i][j]; }

 /*FFTDD*/
 fourn(data,nn,FFT2D,IFFT);

 /*Remplissage*/
 for(i=0;i<(wdth*lgth);i++)
  { ImgFreqR[i]=data[(2*i)+1];
    ImgFreqI[i]=data[(2*i)+2]; }

 /*Conversion en Matrice*/
 for(i=0;i<(wdth*lgth);i++)
  { posy=(int)(i/wdth);
    posx=(int)(i%wdth);

   mtxR[posy][posx]=ImgFreqR[i]/(wdth*lgth);
   mtxI[posy][posx]=ImgFreqI[i]/(wdth*lgth); }

 /*Liberation memoire*/
 free(data);
 free(ImgFreqR);
 free(ImgFreqI);
 free(nn);
}

/*-------------------------------------------------------------*/
/* Mod2                                                        */
/*-------------------------------------------------------------*/
void Mod(float** matM,float** matR,float** matI,int lgth,int wdth)
{
```

```
 int i,j;

 /*Calcul du module*/
 for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
 matM[i][j]=sqrt((matR[i][j]*matR[i][j])+(matI[i][j]*matI[i][j]));
}

/*-------------------------------------------------------------*/
/* Mult                                                        */
/*-------------------------------------------------------------*/
void Mult(float** mat,float coef,int lgth,int wdth)
{
 int i,j;

 /*multiplication*/
  for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
    { mat[i][j]*=coef;
      if (mat[i][j]>GREY_LEVEL) mat[i][j]=GREY_LEVEL; }
}

/*-------------------------------------------------------------*/
/* Recal                                                       */
/*-------------------------------------------------------------*/
void Recal(float** mat,int lgth,int wdth)
{
 int i,j;
 float max,min;

 /*Initialisation*/
 min=mat[0][0];

 /*Recherche du min*/
  for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
    if (mat[i][j]<min) min=mat[i][j];

 /*plus min*/
   for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
    mat[i][j]-=min;

   max=mat[0][0];
 /*Recherche du max*/
  for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
    if (mat[i][j]>max) max=mat[i][j];

 /*Recalibre la matrice*/
 for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
   mat[i][j]*=(GREY_LEVEL/max);
}

/*-------------------------------------------------------------*/
/* Mult 2 matrices complexes                                   */
/*-------------------------------------------------------------*/
void MultMatrix(float** matRout,float** matIout,float** mat1Rin,float** mat1Iin,
float** mat2Rin,float** mat2Iin,int lgth,int wdth)
{
 int i,j;

 for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
   { matRout[i][j]=mat1Rin[i][j]*mat2Rin[i][j]-mat1Iin[i][j]*mat2Iin[i][j];
     matIout[i][j]=mat1Rin[i][j]*mat2Iin[i][j]+mat2Rin[i][j]*mat1Iin[i][j]; }
}

/*-------------------------------------------------------------*/
/* Matrice complexe au carre                                   */
/*-------------------------------------------------------------*/
void SquareMatrix(float** matRout,float** matIout,float** matRin,float** matIin,int lg
th,int wdth)
{
 int i,j;
```

```
 for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
   { matRout[i][j]=SQUARE(matRin[i][j])-SQUARE(matIin[i][j]);
     matIout[i][j]=2*matRin[i][j]*matIin[i][j]; }
}
```