Centro de Investigación y Desarrollo
Tecnológico en Electroquímica, S. C.

PARQUE TECNOLOGICO QUERETARO
SANFANDILA, PEDRO ESCOBEDO
C.P. 76700 - APDO. 064
TEL. Y FAX : (42) 16-57-88 y 16-57-27
E-MAIL: cideteq@sparc.ciateq.conacyt.mx

DESARRO 7 UN SOFTWARE DE ANALISIS DE IMAGENES MED RANSFORMACIONES MORFOLOGICAS

ANEXO
(PRIMITIVAS EN LENGUAJE C)

DR. IVAN TEROL VILLALOBOS
DEPARTAMENTO DE ELECTRONICA Y
AUTOMATIZACION INDUSTRIAL

DEFINICION :

- PARAMETROS GLOBALES.
- ESTRUCTURAS DE DATOS E IMAGENES.
- TRANSFOMACIONES SOBRE LAS IMAGENES.

```
#define VRAI  1
#define FAUX 0
#define BOOLEEN int
#define SIZEIMX 256
#define SIZEIMY 256
#define LEVELG 256
#define LEVELB 2
#define WHITE 255
#define BLACK 0
#define usc unsigned char
#define usi unsigned int
#define NUMIMA 9
#define BINIMA  9
/********************** struct str_image ****************************/
/*IMAGEN ->sx  Taille de l'image sur X                                */
/*IMAGEN ->sy  Taille de l'image sur Y                                */
/*IMAGEN ->lg  Niveaux de gris                                        */
/*IMAGEN ->p[i]  Ligne "i"                                  */
/*SIZEIMX TAILLE MAX. de lignes                             */
/*********************************************************************/
struct str_eimage{
        int sx,sy,lg;
        unsigned int far *p[SIZEIMX+2];
        };
typedef struct str_eimage EIMAGEN;
struct str_image{
        int sx,sy,lg;
        unsigned char far *p[SIZEIMX];
        };
typedef struct str_image IMAGEN;
struct str_bimage{
        int sx,sy,lg;
        unsigned int far *p[SIZEIMX];
        };
typedef struct str_bimage BIMAGEN;
struct noeud{
        int  x;
        int  y;
        struct noeud *suivant;
        };
typedef struct noeud noeud_t;
struct tete{
        int longueur;
        noeud_t *pre, *der;
        };
typedef struct tete tete_t ;
typedef struct tete lacet  ;
```

```
struct fifo{
        int longueur;
        noeud_t *avant,*arriere;
        };
typedef struct fifo fifo_t;
struct fifo_h{
        int long_h;
        fifo_t *file_suivant;
        };
typedef struct fifo_h fifo_h_t;
extern int im_flag;
extern int im_bflag;
extern int gdsmode ;

/************** PRIMIT1.CPP ********************************/
extern void im_greyinv_(IMAGEN *, IMAGEN *);
extern void im_clear_(IMAGEN *);
extern void im_bclear_(BIMAGEN *);
extern int im_rowr_(int ,IMAGEN *,unsigned char *);
extern int im_roww_(int ,IMAGEN *, unsigned char *);
extern int im_columr_(int ,IMAGEN *,unsigned char *);
extern int im_columw_(int ,IMAGEN *, unsigned char *);
extern int readpix(int , int , IMAGEN *);
extern void im_fhisto_(IMAGEN *,unsigned long *);
extern long im_volumen_(IMAGEN *);
extern void im_greysub_const_(IMAGEN *, unsigned char);
extern void im_labelobjet_(BIMAGEN *, IMAGEN *) ;
extern void im_firstobjetbuild_(BIMAGEN *,BIMAGEN *) ;
extern void im_fastbuild_(BIMAGEN *, BIMAGEN *);
extern void im_greybuild_(IMAGEN *, IMAGEN *)    ;
extern void im_distance_(BIMAGEN *, IMAGEN *);
extern usc val_min_min_(usc , usc , usc , usc ,usc );
extern usc val_min_max_(usc ,usc, usc, usc, usc, usc);
extern usc val_set_(usc , usc , usc , usc ,usc );
extern void im_binfillhole_(BIMAGEN *,BIMAGEN *)   ;
extern void im_binedgeoff_(BIMAGEN *,BIMAGEN *) ;
extern void im_binand_(BIMAGEN *,  BIMAGEN *);
extern void im_binor_(BIMAGEN *, BIMAGEN *);
extern void im_binxor_(BIMAGEN *, BIMAGEN *);
extern void im_bindif_(BIMAGEN *,BIMAGEN *) ;
extern void im_bininv_(BIMAGEN *, BIMAGEN *);
extern void im_bcopy_(BIMAGEN *, BIMAGEN *);
extern usc im_breadpix_(BIMAGEN *,int, int) ;
extern void  im_bwritepix_(BIMAGEN *,int, int, usc) ;
extern long im_area_(BIMAGEN *)      ;
extern void im_bshift_(BIMAGEN *, BIMAGEN *, int ,int);
extern void im_bshift_3_(BIMAGEN *, BIMAGEN *, int);
extern void im_bshift_2_(BIMAGEN *, BIMAGEN *, int);
extern void im_bshift_1_(BIMAGEN *, BIMAGEN *, int);
extern void im_bshift_0_(BIMAGEN *, BIMAGEN *, int);
extern void im_copy_(IMAGEN *, IMAGEN *);
extern void im_greysetmask0_(IMAGEN *, BIMAGEN *, IMAGEN *, usc);
extern void im_greysup_(IMAGEN *, IMAGEN *, IMAGEN *);
extern void im_greyinf_(IMAGEN *, IMAGEN *, IMAGEN *);
```

```
extern void im_greysub_(IMAGEN *, IMAGEN *, IMAGEN *);
extern void im_greyadd_(IMAGEN *, IMAGEN *, IMAGEN *);
extern IMAGEN *im_alloc_grey_(int ,int ,int );
extern void im_free_grey_(IMAGEN *);
extern void im_readfileimage_(char *, int , IMAGEN *);
extern void im_writefileimage_(char *,int , usc *, IMAGEN *) ;
extern void im_readheadimage_(char *,int , usc *);
extern void im_readfileimage_g_(char *,int,int,int,int,int,IMAGEN *) ;
extern BIMAGEN *im_alloc_bin_(int ,int ,int );
extern void im_free_bin_(BIMAGEN *);
extern void im_readfilebimage_(char *, int , BIMAGEN *);
extern void im_writefilebimage_(char *,int , usc *, BIMAGEN *) ;
extern void im_readheadbimage_(char *,int , usc *);
/************** PRIMIT0.CPP ******************************/
extern void im_str_ele_(int );
extern void tra_str_ele_();
extern void im_Ngreyerode_(IMAGEN *, IMAGEN *, int );
extern void im_Ngreydilate_(IMAGEN *, IMAGEN *, int );
extern void im_greyerode_m_(IMAGEN *, IMAGEN *);
extern void im_Ngreyopen_(IMAGEN *,IMAGEN *,int );
extern void im_Ngreyclose_(IMAGEN *,IMAGEN *,int );
extern void im_Ngreylindilate_(IMAGEN *, IMAGEN *, int, int) ;
extern void im_greylindilate_(IMAGEN *, int);
extern void im_Ngreylinerode_(IMAGEN *, IMAGEN *, int, int) ;
extern void im_greylinerode_(IMAGEN *, int);
extern void im_Nbinerode_(BIMAGEN *, BIMAGEN *, int );
extern void im_binerode_(BIMAGEN *);
extern void im_Nbindilate_(BIMAGEN *, BIMAGEN *, int );
extern void im_bindilate_(BIMAGEN *);
extern void im_Nbinopen_(BIMAGEN *,BIMAGEN *,int );
extern void im_Nbinclose_(BIMAGEN *,BIMAGEN *,int );
extern void im_Nbindiltri_(BIMAGEN *, BIMAGEN *, int, int) ;
extern void im_bindiltri_(BIMAGEN *, int);
extern void im_Nbinerotri_(BIMAGEN *, BIMAGEN *, int, int) ;
extern void im_binerotri_(BIMAGEN *, int);
extern void im_Nbinopentri_(BIMAGEN *,BIMAGEN *,int,int );
extern void im_Nbinclosetri_(BIMAGEN *,BIMAGEN *,int,int );
extern void im_Nbindillin_(BIMAGEN *, BIMAGEN *, int, int) ;
extern void im_bindillin_(BIMAGEN *, int);
extern void im_Nbinerolin_(BIMAGEN *, BIMAGEN *, int, int) ;
extern void im_binerolin_(BIMAGEN *, int);
extern void im_Nbinopenlin_(BIMAGEN *,BIMAGEN *,int,int );
extern void im_Nbincloselin_(BIMAGEN *,BIMAGEN *,int,int );
/************** PRIMIT2.CPP ******************************/
extern void im_skeleton_(BIMAGEN *, BIMAGEN *, int );
extern void im_quench_(BIMAGEN *,IMAGEN *,int);
extern void im_buildquench_(IMAGEN *,BIMAGEN *);
extern void im_sizedist_bin_(BIMAGEN *, IMAGEN *,float *,int)      ;
extern void im_cova_bin_(BIMAGEN *,float *,int,int,int)      ;
extern void im_Ngradient_(IMAGEN *,IMAGEN *,int, int) ;
extern void im_Nwtophat_(IMAGEN *, IMAGEN *, int );
extern void im_Nbtophat_(IMAGEN *, IMAGEN *, int );
extern void im_fasw_(IMAGEN *, IMAGEN *, int);
extern void im_fasb_(IMAGEN *, IMAGEN *, int);
```

```
extern void im_automed_(IMAGEN *, IMAGEN *, int );
extern void im_centre_sup_(IMAGEN *, IMAGEN *, int);
extern void im_centre_inf_(IMAGEN *,  IMAGEN *, int);
extern BIMAGEN *im_MDFL_0_(int ) ;
/************** VISU.CPP ********************************/
extern void Initialize(void);
extern void ReportStatus(void);
extern void Pause(void);
extern void MainWindow( char *);
extern void StatusLine( char *);
extern void DrawBorder(void);
extern void changetextstyle(int , int , int );
extern int gprintf( int *, int *, char *, ... );
extern void im_greydisplay_(IMAGEN *,int , int , int );
extern void im_bindisplay_(BIMAGEN *,int ,int , int , int );
extern void im_cleardis_(int ,int) ;
extern void im_thresh_(IMAGEN *, BIMAGEN *, usc, usc);
extern void im_adjust_(IMAGEN *, IMAGEN *, usc, usc);
/************** PRIMITI3.CPP ***************************/
extern lacet *cree_lacet_();
extern void im_labeling_(BIMAGEN *, IMAGEN *, IMAGEN *) ;
extern unsigned int val_mask_max_(usi ,usi ,usi ,usi ,usi , usc ) ;
extern lacet *im_lacet_(IMAGEN *,usc ,usc ,usc );
extern noeud_t *inst_noeud_(int ,int ,noeud_t *);
extern fifo_t *cree_file_();
extern BOOLEEN enfile_(int ,int ,fifo_t *);
extern int defiler_(int *,int *,fifo_t *);
extern fifo_h_t *cree_fileh_(int );
extern BOOLEEN free_fileh_(fifo_h_t *);
extern BOOLEEN insere_(int ,int ,tete_t *);
extern int ajoute_(int ,int ,tete_t *);
extern int supprime_(tete_t *);
extern tete_t *cree_liste_();
extern void im_vertientes_(IMAGEN *, IMAGEN *);
extern void im_vertientes_4_(IMAGEN *, IMAGEN *);
extern void im_watershed_(IMAGEN *, IMAGEN *) ;
extern void im_watershed_4_(IMAGEN *, IMAGEN *) ;
```

LIBRERIA : PRIMITO.C

CONTENIDO: TRANSFORMACIONES DE BASE EN MORFOLOGIA MATEMATICA.

1) EROSION MORFOLOGICA UNITARIA USANDO 9 PIXELES COMO ELEMENTO ESTRUCTURANTE (CASO BINARIO Y NUMERICO).

2) DILATACION MORFOLOGICA UNITARIA USANDO 9 PIXELES COMO ELEMENTO ESTRUCTURANTE (CASO BINARIO Y NUMERICO).

3) EROSION Y DILATACION TAMAÑO "N" USANDO 9 PIXELES COMO ELEMENTO ESTRUCTURANTE (CASO BINARIO Y NUMERICO).

4) APERTURA Y CERRADURA MORFOLOGICA USANDO 9 PIXELES COMO ELEMENTO ESTRUCTURANTE (BINARIO Y NUMERICO).

5) EROSION Y DILATACION LINEAL (8 DIRECCIONES) USANDO DOS PIXELES VECINOS COMO ELEMENTO ESTRUCTURANTE.

6) APERTURA Y CERRADURA LINEAL (8 DIRECCIONES) USANDO DOS PIXELES VECINOS COMO ELEMENTO ESTRUCTURANTE.

7) EROSION Y DILATACION TRIANGULAR (8 DIRECCIONES) USANDO TRES PIXELES VECINOS COMO ELEMENTO ESTRUCTURANTE.

8) APERTURA Y CERRADURA TRIANGULAR (8 DIRECCIONES) USANDO TRES PIXELES VECINOS COMO ELEMENTO ESTRUCTURANTE.

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include "c:\users\ivan\logiciel\strim.h"

int gdsmode    ;
extern int im_flag       ;

/*-------------------------------------------------------------------------------------------------------------*/
/*       TRANSF: EROSION MORFOLOGICA NUMERICA TAMAÑO "N"        */
/*        ENTRADAS:    ime  ESTRUCTURA IMAGEN NUMERICA          */
/*                     size TAMAÑO DE LA EROSION                */
/*       SALIDAS:      ims ESTRUCTURA IMAGEN NUMERICA           */
/*-------------------------------------------------------------------------------------------------------------*/

void im_Ngreyerode_(IMAGEN *ime, IMAGEN *ims, int size)
{
        int i    ;
```

```
        for(i=0 ; i<(ims->sx+2) ;i++){
                ims->p[0][i]= WHITE             ;
                ims->p[ims->sy+1][i]=WHITE      ;
                ims->p[i][0]=WHITE              ;
                ims->p[i][ims->sx+1]=WHITE      ;
        }

        if(ime!=ims)
                im_copy_(ime,ims)       ;
        for(i=0 ; i<size ; i++)
                im_greyerode_(ims)      ;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION NUMERICA LINEAL TAMAÑO "N"                                      */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA                                    */
/*                   size TAMAÑO DE LA EROSION                                          */
/*                   dir   DIRECCION                                                    */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                                     */
/*-------------------------------------------------------------------------------------------------------------*/


void im_Ngreylinerode_(IMAGEN *ime, IMAGEN *ims, int size, int dir)
{
        int i   ;


        for(i=0 ; i<(ims->sx+2) ;i++){
                ims->p[0][i]= WHITE             ;
                ims->p[ims->sy+1][i]=WHITE      ;
                ims->p[i][0]=WHITE              ;
                ims->p[i][ims->sx+1]=WHITE      ;
        }

        if(ime!=ims)
                im_copy_(ime,ims)       ;
        for(i=0 ; i<size ; i++)
                im_greylinerode_(ims,dir)       ;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION NUMERICA LINEAL TAMAÑO "N"                                      */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA                                    */
/*                   size TAMAÑO DE LA EROSION                                          */
/*                   dir   DIRECCION                                                    */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                                     */
/*-------------------------------------------------------------------------------------------------------------*/


void im_Ngreylindilate_(IMAGEN *ime, IMAGEN *ims, int size, int dir)
{
        int i   ;
```

```
        if(ime! = ims)
                im_copy_(ime,ims)       ;

        for(i = 0 ; i < (ims->sx + 2) ;i + +){
                ims->p[0][i] =  BLACK           ;
                ims->p[ims->sy + 1][i] = BLACK          ;
                ims->p[i][0] = BLACK            ;
                ims->p[i][ims->sx + 1] = BLACK          ;
        }

        for(i = 0 ; i < size ; i + +)
                im_greylindilate_(ims,dir)              ;
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION NUMERICA LINEAL UNITARIA                                        */
/*       ENTRADAS:    ims  ESTRUCTURA IMAGEN NUMERICA                                   */
/*                    dir   DIRECCION                                                   */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN NUMERICA                                    */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_greylinerode_(IMAGEN *ims, int dir)
{

        int i,j                         ;

switch(dir){
case 0:
for(i = 1 ; i < = ims->sy ;i + +)
        for(j = 1 ; j < = ims->sx ; j + +)
        if((usc)ims->p[i][j] > (usc)ims->p[i][j + 1])ims->p[i][j] = ims->p[i][j + 1];
        break ;

case 1:
for(i = ims->sy ; i > = 1 ;i--)
        for(j = ims->sx ; j > = 1 ; j--)
                if((usc)ims->p[i][j] > (usc)ims->p[i-1][j])ims->p[i][j] = ims->p[i-1][j]    ;
                break ;
case 2:
for(i = 1 ; i < = ims->sy ;i + +)
        for(j = ims->sx ; j > = 1 ; j--)
                if((usc)ims->p[i][j] > (usc)ims->p[i][j-1])ims->p[i][j] = ims->p[i][j-1]    ;
                break ;
case 3:
for(i = 1 ; i < = ims->sy ;i + +)
        for(j = 1 ; j < = ims->sx ; j + +)
                if((usc)ims->p[i][j] > (usc)ims->p[i + 1][j])ims->p[i][j] = ims->p[i + 1][j];
                break ;
        default : printf("\n *** Erreur d'option (0-3)") ;
}


}
```

```
/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: DILATACION MORFOLOGICA NUMERICA TAMAÑO "N"    */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA          */
/*                   size TAMAÑO DE LA EROSION                */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA           */
/*-------------------------------------------------------------------------------------------------------------*/


void im_Ngreydilate_(IMAGEN *ime, IMAGEN *ims, int size)
{
        int i    ;

        if(ime! =ims)
                im_copy_(ime,ims)      ;

        for(i=0 ; i<(ims->sx+2) ;i++){
                ims->p[0][i]= BLACK          ;
                ims->p[ims->sy+1][i]=BLACK       ;
                ims->p[i][0]=BLACK           ;
                ims->p[i][ims->sx+1]=BLACK       ;
        }

        for(i=0 ; i<size ; i++)
                im_greydilate_(ims)          ;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: DILATACION MORFOLOGICA NUMERICA UNITARIA      */
/*       ENTRADAS:   ims  ESTRUCTURA IMAGEN NUMERICA          */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA           */
/*-------------------------------------------------------------------------------------------------------------*/

void im_greydilate_(IMAGEN *ims)
{

        int i,j                         ;

for(i=1 ; i<=ims->sy ;i++)
        for(j=1 ; j<=ims->sx ; j++)
                if((usc)ims->p[i][j]<(usc)ims->p[i][j+1])ims->p[i][j]=ims->p[i][j+1];

for(i=ims->sy ; i>=1 ;i--)
        for(j=ims->sx ; j>=1 ; j--)
                if((usc)ims->p[i][j]<(usc)ims->p[i-1][j])ims->p[i][j]=ims->p[i-1][j]    ;

for(i=1 ; i<=ims->sy ;i++)
        for(j=ims->sx ; j>=1 ; j--)
                if((usc)ims->p[i][j]<(usc)ims->p[i][j-1])ims->p[i][j]=ims->p[i][j-1]    ;

for(i=1 ; i<=ims->sy ;i++)
        for(j=1 ; j<=ims->sx ; j++)
                if((usc)ims->p[i][j]<(usc)ims->p[i+1][j])ims->p[i][j]=ims->p[i+1][j];
```

```
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: DILATACION NUMERICA LINEAL UNITARIA                                                                  */
/*       ENTRADAS:   ims  ESTRUCTURA IMAGEN NUMERICA                                                                 */
/*                   dir  DIRECCION                                                                                  */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                                                                 */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_greylindilate_(IMAGEN *ims, int dir)
{

        int i,j                         ;

switch(dir){
case 0 :
for(i=1 ; i<=ims->sy ;i++)
        for(j=1 ; j<=ims->sx ; j++)
                if((usc)ims->p[i][j]<(usc)ims->p[i][j+1])ims->p[i][j]=ims->p[i][j+1];
                break;
case 1:
for(i=ims->sy ; i>=1 ;i--)
        for(j=ims->sx ; j>=1 ; j--)
                if((usc)ims->p[i][j]<(usc)ims->p[i-1][j])ims->p[i][j]=ims->p[i-1][j]    ;
                break ;
case 2:
for(i=1 ; i<=ims->sy ;i++)
        for(j=ims->sx ; j>=1 ; j--)
                if((usc)ims->p[i][j]<(usc)ims->p[i][j-1])ims->p[i][j]=ims->p[i][j-1]    ;
                break ;
case 3:

for(i=1 ; i<=ims->sy ;i++)
        for(j=1 ; j<=ims->sx ; j++)
                if((usc)ims->p[i][j]<(usc)ims->p[i+1][j])ims->p[i][j]=ims->p[i+1][j];
                break ;
        default : printf("\n *** Erreur d'option (0-3)") ;
        }

}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION MORFOLOGICA NUMERICA UNITARIA                                                                */
/*       ENTRADAS:   ims  ESTRUCTURA IMAGEN NUMERICA                                                                 */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                                                                 */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_greyerode_(IMAGEN *ims)
{

        int i,j                         ;

/******************   EROSION          ***************/
```

```
for(i=1 ; i<=ims->sy ;i++)
        for(j=1 ; j<=ims->sx ; j++)
        if((usc)ims->p[i][j]>(usc)ims->p[i][j+1])ims->p[i][j]=ims->p[i][j+1];


for(i=ims->sy ; i>=1 ;i--)
        for(j=ims->sx ; j>=1 ; j--)
                if((usc)ims->p[i][j]>(usc)ims->p[i-1][j])ims->p[i][j]=ims->p[i-1][j]    ;

for(i=1 ; i<=ims->sy ;i++)
        for(j=ims->sx ; j>=1 ; j--)
                if((usc)ims->p[i][j]>(usc)ims->p[i][j-1])ims->p[i][j]=ims->p[i][j-1]    ;

for(i=1 ; i<=ims->sy ;i++)
        for(j=1 ; j<=ims->sx ; j++)
                if((usc)ims->p[i][j]>(usc)ims->p[i+1][j])ims->p[i][j]=ims->p[i+1][j];

}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: APERTURA MORFOLOGICA NUMERICA TAMAÑO "N"        */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA            */
/*                   size TAMAÑO DE LA EROSION                  */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA             */
/*-------------------------------------------------------------------------------------------------------------*/


void im_Ngreyopen_(IMAGEN *ime,IMAGEN *ims,int size)
{
        im_Ngreyerode_(ime,ims,size)  ;
        im_Ngreydilate_(ims,ims,size)  ;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CERRADURA MORFOLOGICA NUMERICA TAMAÑO "N"       */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA            */
/*                   size TAMAÑO DE LA EROSION                  */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA             */
/*-------------------------------------------------------------------------------------------------------------*/

void im_Ngreyclose_(IMAGEN *ime,IMAGEN *ims,int  size)
{
        im_Ngreydilate_(ime,ims,size)  ;
        im_Ngreyerode_(ims,ims,SIZe) ;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: APERTURA MORFOLOGICA BINARIA TAMAÑO "N"         */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARI A            */
/*                   size TAMAÑO DE LA APERTURA                 */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA              */
/*-------------------------------------------------------------------------------------------------------------*/
```

```
void im_Nbinopen_(BIMAGEN *ime,BIMAGEN *ims,int taille)
{
        im_Nbinerode_(ime,ims,taille)  ;
        im_Nbindilate_(ims,ims,taille)  ;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CERRADURA MORFOLOGICA BINARIA TAMAÑO "N"        */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA             */
/*                   size TAMAÑO DE LA CERRADURA                */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA              */
/*-------------------------------------------------------------------------------------------------------------*/


void im_Nbinclose_(BIMAGEN *ime,BIMAGEN *ims,int size)
{
        im_Nbindilate_(ime,ims,size)   ;
        im_Nbinerode_(ims,ims,size)   ;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CERRADURA MORFOLOGICA BINARIA TAMAÑO "N"        */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA             */
/*                   size TAMAÑO DE LA EROSION                  */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA              */
/*-------------------------------------------------------------------------------------------------------------*/


void im_Nbinerode_(BIMAGEN *ime, BIMAGEN *ims, int size)
{
        int i    ;

        if(ime! =ims)
                im_bcopy_(ime,ims)    ;

        for(i=0 ; i<size ; i++)
                im_binerode_(ims)      ;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION MORFOLOGICA  BINARIA UNITARIA           */
/*       ENTRADAS:   ims  ESTRUCTURA IMAGEN BINARIA             */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA              */
/*-------------------------------------------------------------------------------------------------------------*/

void im_binerode_(BIMAGEN *ims)
{

        int i,j                         ;
        unsigned int vmask0,vmask1   ;
        for(i=0  ; i<ims->sy  ; i++){
                if(gdsmode= =1)
                        vmask0=0x8000       ;
```

```
                else
                        vmask0 = 0x0000         ;
                for(j=0  ;j<(ims->sx/16)  ; j++ ){
                vmask1 =0x0001 & ims->p[i][j]       ;
                vmask1 =((0x0001 & ims->p[i][j]) ==0x0001) ? 0x8000 : 0 ;
                ims->p[i][j] &= ((ims->p[i][j] >> 1 ) | vmask0)      ;
                vmask0=vmask1       ;
                }
        }
        for(i=0 ; i<(ims->sy-1) ;i++)
                for(j=0 ; j<(ims->sx/16) ;j++)
                        ims->p[i][j] &= ims->p[i+1][j]          ;
                if(gdsmode==1)
                        for(j=0 ; j<(ims->sx/16) ;j++)
                                ims->p[ims->sy-1][j] &= 0xFFFF      ;
                else
                        for(j=0 ; j<(ims->sx/16) ; j++)
                                ims->p[ims->sy-1][j]=0       ;

        for(i=0  ; i<ims->sy  ; i++){
                if(gdsmode==1)
                        vmask0=0x00001     ;
                else
                        vmask0 = 0x0000         ;
                for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
                vmask1 =((0x8000 & ims->p[i][j]) ==0x8000) ? 1 : 0 ;
                ims->p[i][j] &= ((ims->p[i][j] << 1 ) | vmask0)      ;
                vmask0=vmask1       ;
                }
        }

        for(i=(ims->sy-1) ;i>0 ;i--)
                for(j=0 ; j<(ims->sx/16) ;j++)
                        ims->p[i][j] &= ims->p[i-1][j] ;
        if(gdsmode==1)
                for(j=0 ; j<(ims->sx/16) ; j++)
                        ims->p[0][j] &=0xFFFF        ;
        else
                for(j=0 ; j<(ims->sx/16) ; j++)
                        ims->p[0][j]=0         ;


}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: DILATACION MORFOLOGICA BINARIA TAMAÑO "N"       */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA             */
/*                   size TAMAÑO DE LA DILATACION               */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA              */
/*-------------------------------------------------------------------------------------------------------------*/


void im_Nbindilate_(BIMAGEN *ime, BIMAGEN *ims, int size)
{
```

```
        int i     ;

        if(ime! =ims)
                im_bcopy_(ime,ims)     ;

        for(i=0 ; i<size ; i++){
                im_bindilate_(ims)       ; }
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: DILATACION MORFOLOGICA BIMARIA UNITARIA                 */
/*       ENTRADAS:    ims  ESTRUCTURA IMAGEN BINARIA                    */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN BINARIA                    */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_bindilate_(BIMAGEN *ims)
{

        int i,j                          ;
        unsigned int vmask0,vmask1   ;

        for(i=0  ; i<ims->sy  ; i++){
                if(gdsmode = =1)
                        vmask0=0x8000       ;
                else
                        vmask0  =  0x0000          ;
                for(j=0  ;j<(ims->sx/16)  ; j++ ){
                vmask1=0x0001 & ims->p[i][j]        ;
                vmask1=((0x0001 & ims->p[i][j]) = =0x0001) ? 0x8000 : 0 ;
                ims->p[i][j] | = ((ims->p[i][j] > > 1 ) | vmask0)       ;
                vmask0=vmask1       ;
                }
        }
        for(i=0 ; i<(ims->sy-1) ;i++)
                for(j=0 ; j<(ims->sx/16) ;j++)
                        ims->p[i][j] | = ims->p[i+1][j]          ;
                if(gdsmode = =1)
                        for(j=0 ; j<(ims->sx/16) ;j++)
                                ims->p[ims->sy-1][j] & = 0xFFFF      ;
                else
                for(j=0 ; j<(ims->sx/16) ; j++)
                        ims->p[ims->sy-1][j]=0       ;

        for(i=0  ; i<ims->sy  ; i++){
                if(gdsmode = =1)
                        vmask0=0x00001     ;
                else
                        vmask0  =  0x0000          ;
                for(j=(ims->sx/16)-1 ; j> =0  ; j-- ){
                vmask1=((0x8000 & ims->p[i][j]) = =0x8000) ? 1 : 0 ;
                ims->p[i][j] | = ((ims->p[i][j] < < 1 ) | vmask0)       ;
                vmask0=vmask1       ;
                }
        }
```

```
        for(i=(ims->sy-1) ;i>0 ;i--)
                for(j=0 ; j<(ims->sx/16) ;j++)
                        ims->p[i][j] |= ims->p[i-1][j] ;
        if(gdsmode==1)
                for(j=0 ; j<(ims->sx/16) ; j++)
                        ims->p[0][j] &=0xFFFF       ;
        else
                for(j=0 ; j<(ims->sx/16) ; j++)
                        ims->p[0][j]=0          ;


}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: DILATACION BINARIA TRIANGULAR TAMAÑO "N"                */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA                     */
/*                   size TAMAÑO DE LA DILATACION                       */
/*                   dir  DIRECCION                                     */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA                      */
/*-------------------------------------------------------------------------------------------------------------*/

void im_Nbindiltri_(BIMAGEN *ime, BIMAGEN *ims, int size, int dir)
{
        int i    ;

        if(ime!=ims)
                im_bcopy_(ime,ims)    ;

        for(i=0 ; i<size ; i++){
                im_bindiltri_(ims,dir)    ; }
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: DILATACION BINARIA TRIANGULAR UNITARIA                  */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA                     */
/*                   dir  DIRECCION                                     */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA                      */
/*-------------------------------------------------------------------------------------------------------------*/

void im_bindiltri_(BIMAGEN *ims, int dir)
{
/*********************************************************/
/*   Element Structurante                        */
/* 0  °'  1 'o  2 °  3   °   ' Centre     */
/*     o     o      'o   o '            */
/*********************************************************/

        int i,j                         ;
        unsigned int vmask0,vmask1  ;
        BIMAGEN *imaux                      ;

        imaux = im_alloc_bin_(SIZEIMX,SIZEIMY,LEVELB)     ;
```

```
switch(dir){
case 0 :
for(i=0  ; i<ims->sy  ; i++){
        vmask0  =  0x0000          ;
        for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
        imaux->p[i][j]  =  ims->p[i][j]   ;
        vmask1=((0x8000 & ims->p[i][j])==0x8000) ? 1 : 0 ;
        imaux->p[i][j] |= ((ims->p[i][j] << 1 ) | vmask0)    ;
        vmask0=vmask1        ;
        }
}
for(i=(ims->sy-1) ;i>0 ;i--)
        for(j=0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j]  =  imaux->p[i][j] | ims->p[i-1][j]  ;
for(j=0 ; j<(ims->sx/16) ; j++)
        ims->p[0][j]=0          ;
        break ;
case 1 :
for(i=0  ; i<ims->sy  ; i++){
        vmask0  =  0x0000          ;
        for(j=0  ;j<(ims->sx/16)  ; j++ ){
        imaux->p[i][j]  =  ims->p[i][j]   ;
        vmask1=((0x0001 & ims->p[i][j])==0x0001) ? 0x8000 : 0 ;
        imaux->p[i][j] |= ((ims->p[i][j] >> 1 ) | vmask0)    ;
        vmask0=vmask1        ;
        }
}
for(i=(ims->sy-1) ;i>0 ;i--)
        for(j=0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j]  =  imaux->p[i][j] | ims->p[i-1][j]  ;
for(j=0 ; j<(ims->sx/16) ; j++)
        ims->p[0][j]=0          ;
        break ;
case 2 :
for(i=0  ; i<ims->sy  ; i++){
        vmask0  =  0x0000          ;
        for(j=0  ;j<(ims->sx/16)  ; j++ ){
        imaux->p[i][j]  =  ims->p[i][j]   ;
        vmask1=((0x0001 & ims->p[i][j])==0x0001) ? 0x8000 : 0 ;
        imaux->p[i][j] |= ((ims->p[i][j] >> 1 ) | vmask0)    ;
        vmask0=vmask1        ;
        }
}
for(i=0 ; i<(ims->sy-1) ;i++)
        for(j=0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j]  =  imaux->p[i][j] | ims->p[i+1][j];
for(j=0 ; j<(ims->sx/16) ; j++)
        ims->p[ims->sy-1][j]=0       ;
                break ;
case 3 :
for(i=0  ; i<ims->sy  ; i++){
        vmask0  =  0x0000          ;
        for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
```

```
                imaux->p[i][j] = ims->p[i][j]   ;
                vmask1 = ((0x8000 & ims->p[i][j]) = = 0x8000) ? 1 : 0 ;
                imaux->p[i][j] | = ((ims->p[i][j] < < 1 ) | vmask0)    ;
                vmask0 = vmask1         ;
                }
        }
        for(i = 0 ; i<(ims->sy-1) ;i + + )
                for(j = 0 ; j<(ims->sx/16) ;j + + )
                        ims->p[i][j] = imaux->p[i][j] | ims->p[i + 1][j];
        for(j = 0 ; j<(ims->sx/16) ; j + + )
                ims->p[ims->sy-1][j] = 0         ;
                        break ;
        default : printf("\n *** Erreur d'option (0-3)") ;
        }
        im_free_bin_(imaux)     ;

}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION BINARIA TRIANGULAR TAMAÑO "N"                   */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN BINARIA                    */
/*                    size TAMAÑO DE LA EROSION                         */
/*                    dir   DIRECCION                                   */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA                     */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_Nbinerotri_(BIMAGEN *ime, BIMAGEN *ims, int size, int dir)
{
        int i     ;

        if(ime! = ims)
                im_bcopy_(ime,ims)     ;

        for(i = 0 ; i<size ; i + + ){
                im_binerotri_(ims, dir)  ; }
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION  BINARIA TRIANGULAR UNITARIA                    */
/*       ENTRADAS:    ims  ESTRUCTURA IMAGEN BINARIA                    */
/*                    dir   DIRECCION                                   */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA                     */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_binerotri_(BIMAGEN *ims, int dir)
{
/*********************************************************/
/*   Element Structurante                        */
/* 0   º '   1 ' º   2 º   3    º    ' Centre     */
/*      º     º       ' º    º '              */
/*********************************************************/

        int i,j                         ;
        unsigned int vmask0,vmask1   ;
```

```
BIMAGEN *imaux                  ;

imaux = im_alloc_bin_(SIZEIMX,SIZEIMY,LEVELB)    ;

switch(dir){
case 0 :
for(i=0  ; i<ims->sy  ; i++){
        vmask0 = 0x0000         ;
        for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
        imaux->p[i][j] = ims->p[i][j]  ;
        vmask1 =((0x8000 & ims->p[i][j])==0x8000) ? 1 : 0 ;
        imaux->p[i][j] &= ((ims->p[i][j] << 1 ) | vmask0)   ;
        vmask0=vmask1       ;
        }
}
for(i=(ims->sy-1) ;i>0 ;i--)
        for(j=0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j] = imaux->p[i][j] & ims->p[i-1][j] ;
for(j=0 ; j<(ims->sx/16) ; j++)
        ims->p[0][j]=0          ;
        break ;
case 1 :
for(i=0  ; i<ims->sy  ; i++){
        vmask0 = 0x0000         ;
        for(j=0  ;j<(ims->sx/16)  ; j++ ){
        imaux->p[i][j] = ims->p[i][j]  ;
        vmask1 =((0x0001 & ims->p[i][j])==0x0001) ? 0x8000 : 0 ;
        imaux->p[i][j] &= ((ims->p[i][j] >> 1 ) | vmask0)   ;
        vmask0=vmask1       ;
        }
}
for(i=(ims->sy-1) ;i>0 ;i--)
        for(j=0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j] = imaux->p[i][j] & ims->p[i-1][j] ;
for(j=0 ; j<(ims->sx/16) ; j++)
        ims->p[0][j]=0          ;
        break ;
case 2 :
for(i=0  ; i<ims->sy  ; i++){
        vmask0 = 0x0000         ;
        for(j=0  ;j<(ims->sx/16)  ; j++ ){
        imaux->p[i][j] = ims->p[i][j]  ;
        vmask1 =((0x0001 & ims->p[i][j])==0x0001) ? 0x8000 : 0 ;
        imaux->p[i][j] &= ((ims->p[i][j] >> 1 ) | vmask0)   ;
        vmask0=vmask1       ;
        }
}
for(i=0 ; i<(ims->sy-1) ;i++)
        for(j=0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j] = imaux->p[i][j] & ims->p[i+1][j]          ;
for(j=0 ; j<(ims->sx/16) ; j++)
        ims->p[ims->sy-1][j]=0      ;
```

```
                        break ;
        case 3 :
        for(i=0  ; i<ims->sy  ; i++){
                vmask0  =  0x0000          ;
                for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
                imaux->p[i][j]  =  ims->p[i][j]   ;
                vmask1 =((0x8000 & ims->p[i][j])==0x8000) ? 1 : 0 ;
                imaux->p[i][j] &= ((ims->p[i][j] << 1 ) | vmask0)    ;
                vmask0=vmask1          ;
                }
        }
        for(i=0 ; i<(ims->sy-1) ;i++)
                for(j=0 ; j<(ims->sx/16) ;j++)
                        ims->p[i][j]  =  imaux->p[i][j] & ims->p[i+1][j]          ;
        for(j=0 ; j<(ims->sx/16) ; j++)
                ims->p[ims->sy-1][j]=0          ;
                        break ;
        default : printf("\n *** Erreur d'option (0-3)") ;
        }

        im_free_bin_(imaux)     ;

}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: APERTURA BINARIA TRIANGULAR TAMAÑO "N"                  */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN BINARIA                    */
/*                    size TAMAÑO DE LA APERTURA                        */
/*                    dir   DIRECCION                                   */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA                     */
/*-------------------------------------------------------------------------------------------------------------------*/


void im_Nbinopentri_(BIMAGEN *ime,BIMAGEN *ims,int taille, int dir)
{
        im_Nbinerotri_(ime,ims,taille,dir)          ;
        im_Nbindiltri_(ims,ims,taille,(dir+2)%4)          ;
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CERRADURA BINARIA TRIANGULAR TAMAÑO "N"                 */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN BINARIA                    */
/*                    size TAMAÑO DE LA CERRADURA                       */
/*                    dir   DIRECCION                                   */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA                     */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_Nbinclosetri_(BIMAGEN *ime,BIMAGEN *ims,int taille,int dir)
{
        im_Nbindiltri_(ime,ims,taille,dir)          ;
        im_Nbinerotri_(ims,ims,taille,(dir+2)%4)          ;
}

/*-------------------------------------------------------------------------------------------------------------------*/
```

```
/*      TRANSF: DILATACION BINARIA LINEAR TAMAÑO "N"                    */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA                     */
/*                   size TAMAÑO DE LA DILATACION                       */
/*                   dir   DIRECCION                                    */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA                      */
/*-------------------------------------------------------------------------*/

void im_Nbindillin_(BIMAGEN *ime, BIMAGEN *ims, int size, int dir)
{
        int i    ;

        if(ime! =ims)
                im_bcopy_(ime,ims)    ;

        for(i=0 ; i<size ; i++){
                im_bindillin_(ims,dir)    ; }
}

/*-------------------------------------------------------------------------*/
/*      TRANSF: DILATACION BINARIA LINEAR UNITARIA                      */
/*       ENTRADAS:   ims  ESTRUCTURA IMAGEN BINARIA                     */
/*                   dir   DIRECCION                                    */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA                      */
/*-------------------------------------------------------------------------*/

void im_bindillin_(BIMAGEN *ims, int dir)
{

/*********************************************************/
/*   Element Structurante                        */
/* 0   º '   1  º   2  ' º   3  '   ( ' Centre)     */
/*           '            º                */
/*********************************************************/


        int i,j                         ;
        unsigned int vmask0,vmask1   ;


        switch(dir){
        case 0 :
        for(i=0  ; i<ims->sy  ; i++){
                vmask0  =  0x0000          ;
                for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
                vmask1 =((0x8000 & ims->p[i][j])==0x8000) ? 1 : 0 ;
                ims->p[i][j] |= ((ims->p[i][j] << 1 ) | vmask0)       ;
                vmask0=vmask1        ;
                }
        }
                break ;
        case 1 :
        for(i=0  ; i<(ims->sy-1)  ; i++){
                vmask0  =  0x0000          ;
                for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
```

```
        vmask1 =((0x8000 & ims->p[i+1][j]) ==0x8000) ? 1 : 0 ;
        ims->p[i][j] |= ((ims->p[i+1][j] << 1 ) | vmask0)   ;
        vmask0 = vmask1        ;
        }
}
        break ;
case 2 :
for(i =(ims->sy-1) ;i>0 ;i--)
        for(j =0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j] |= ims->p[i-1][j] ;
for(j =0 ; j<(ims->sx/16) ; j++)
        ims->p[0][j] =0          ;
        break ;
case 3 :
for(i =0  ; i<(ims->sy-1)  ; i++){
        vmask0  =  0x0000          ;
        for(j =0  ;j<(ims->sx/16)  ; j++ ){
        vmask1 =((0x0001 & ims->p[i+1][j]) ==0x0001) ? 0x8000 : 0 ;
        ims->p[i][j] |= ((ims->p[i+1][j] >> 1 ) | vmask0)   ;
        vmask0 = vmask1        ;
        }
}
                break ;
case 4 :
for(i =0  ; i<ims->sy  ; i++){
        vmask0  =  0x0000          ;
        for(j =0  ;j<(ims->sx/16)  ; j++ ){
        vmask1 =((0x0001 & ims->p[i][j]) ==0x0001) ? 0x8000 : 0 ;
        ims->p[i][j] |= ((ims->p[i][j] >> 1 ) | vmask0)        ;
        vmask0 = vmask1        ;
        }
}
                break ;
case 5 :
for(i =(ims->sy-1); i>0  ; i--){
        vmask0  =  0x0000          ;
        for(j =0  ;j<(ims->sx/16)  ; j++ ){
        vmask1 =((0x0001 & ims->p[i-1][j]) ==0x0001) ? 0x8000 : 0 ;
        ims->p[i][j] |= ((ims->p[i-1][j] >> 1 ) | vmask0)     ;
        vmask0 = vmask1        ;
        }
}
                break ;
case 6 :
for(i =0 ; i<(ims->sy-1) ;i++)
        for(j =0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j] |= ims->p[i+1][j]          ;
for(j =0 ; j<(ims->sx/16) ; j++)
        ims->p[ims->sy-1][j] =0        ;
                break ;
case 7 :
for(i =(ims->sy-1); i>0  ; i--){
        vmask0  =  0x0000          ;
        for(j =(ims->sx/16)-1 ; j> =0  ; j-- ){
```

```
                vmask1 = ((0x8000 & ims->p[i-1][j]) = = 0x8000) ? 1 : 0 ;
                ims->p[i][j] | = ((ims->p[i-1][j] < < 1 ) | vmask0)     ;
                vmask0 = vmask1         ;
                }
        }
                break ;
        default : printf("\n *** Erreur d'option (0-7)") ;
        }



}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION     BINARIA LINEAR TAMAÑO "N"                   */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN BINARIA                   */
/*                    size TAMAÑO DE LA EROSION                        */
/*                    dir  DIRECCION                                   */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA                    */
/*-------------------------------------------------------------------------------------------------------------*/

void im_Nbinerolin_(BIMAGEN *ime, BIMAGEN *ims, int size, int dir)
{
        int i     ;

        if(ime! = ims)
                im_bcopy_(ime,ims)     ;

        for(i = 0 ; i<size ; i + + ){
                im_binerolin_(ims, dir)  ; }
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: EROSION BINARIA LINEAR UNITARIA                         */
/*       ENTRADAS:    ims  ESTRUCTURA IMAGEN BINARIA                   */
/*                    dir  DIRECCION                                   */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA                    */
/*-------------------------------------------------------------------------------------------------------------*/

void im_binerolin_(BIMAGEN *ims, int dir)
{
/*********************************************************/
/*   Element Structurante                    */
/* 0   °'   2  °   4  ' °   6  '   ( ' Centre)     */
/*          '           °                */
/*********************************************************/

        int i,j                         ;
        unsigned int vmask0,vmask1   ;
        switch(dir){
        case 0 :
        for(i = 0  ; i<ims->sy  ; i + + ){
                vmask0  =  0x0000         ;
```

```
        for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
        vmask1 =((0x8000 & ims->p[i][j]) ==0x8000) ? 1 : 0 ;
        ims->p[i][j] &= ((ims->p[i][j] << 1 ) | vmask0)      ;
        vmask0=vmask1       ;
        }
}
        break ;
case 1 :
for(i=0  ; i<(ims->sy-1)  ; i++){
        vmask0  =  0x0000          ;
        for(j=(ims->sx/16)-1 ; j>=0  ; j-- ){
        vmask1 =((0x8000 & ims->p[i+1][j]) ==0x8000) ? 1 : 0 ;
        ims->p[i][j] &= ((ims->p[i+1][j] << 1 ) | vmask0)  ;
        vmask0=vmask1       ;
        }
}
for(j=(ims->sx/16)-1 ; j>=0 ; j--)
        ims->p[ims->sy-1][j]=0       ;
        break ;
case 2 :
for(i=(ims->sy-1) ;i>0 ;i--)
        for(j=0 ; j<(ims->sx/16) ;j++)
                ims->p[i][j] &= ims->p[i-1][j] ;
for(j=0 ; j<(ims->sx/16) ; j++)
        ims->p[0][j]=0          ;
        break ;
case 3 :
for(i=0  ; i<(ims->sy-1)  ; i++){
        vmask0  =  0x0000          ;
        for(j=0  ;j<(ims->sx/16)  ; j++ ){
        vmask1 =((0x0001 & ims->p[i+1][j]) ==0x0001) ? 0x8000 : 0 ;
        ims->p[i][j] &= ((ims->p[i+1][j] >> 1 ) | vmask0)  ;
        vmask0=vmask1       ;
        }
}
for(j=0 ; j<ims->sx/16 ; j++)
        ims->p[ims->sy-1][j]=0       ;
                break ;
case 4 :
for(i=0  ; i<ims->sy  ; i++){
        vmask0  =  0x0000          ;
        for(j=0  ;j<(ims->sx/16)  ; j++ ){
        vmask1 =((0x0001 & ims->p[i][j]) ==0x0001) ? 0x8000 : 0 ;
        ims->p[i][j] &= ((ims->p[i][j] >> 1 ) | vmask0)      ;
        vmask0=vmask1       ;
        }
}
                break ;
case 5 :
for(i=(ims->sy-1); i>0  ; i--){
        vmask0  =  0x0000          ;
        for(j=0  ;j<(ims->sx/16)  ; j++ ){
        vmask1 =((0x0001 & ims->p[i-1][j]) ==0x0001) ? 0x8000 : 0 ;
        ims->p[i][j] &= ((ims->p[i-1][j] >> 1 ) | vmask0)  ;
```

```
                vmask0 = vmask1         ;
                }
        }
        for(j = 0 ; j < ims->sx/16 ; j + + )
                ims->p[0][j] = 0        ;
                        break ;
        case_6 :
        for(i = 0 ; i < (ims->sy-1) ;i + + )
                for(j = 0 ; j < (ims->sx/16) ;j + + )
                        ims->p[i][j] &= ims->p[i + 1][j]        ;
        for(j = 0 ; j < (ims->sx/16) ; j + + )
                ims->p[ims->sy-1][j] = 0        ;
                        break ;
        case 7 :
        for(i = (ims->sy-1); i > 0  ; i--){
                vmask0  =  0x0000       ;
                for(j = (ims->sx/16)-1 ; j > = 0  ; j-- ){
                vmask1 = ((0x8000 & ims->p[i-1][j]) = = 0x8000) ? 1 : 0 ;
                ims->p[i][j] &= ((ims->p[i-1][j] < < 1 ) | vmask0)    ;
                vmask0 = vmask1         ;
                }
        }
        for(j = (ims->sx/16)-1 ; j > = 0 ; j--)
                ims->p[0][j] = 0        ;
                break ;
        default : printf("\n *** Erreur d'option (0-7)") ;
        }



}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: APERTURA BINARIA LINEAR TAMAÑO "N"          */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA                 */
/*                   size TAMAÑO DE LA APERTURA                     */
/*                   dir   DIRECCION                                */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA                  */
/*-------------------------------------------------------------------------------------------------------------------*/


void im_Nbinopenlin_(BIMAGEN *ime,BIMAGEN *ims,int taille,int dir)
{
        im_Nbinerolin_(ime,ims,taille,dir)      ;
        im_Nbindillin_(ims,ims,taille,(dir + 4)%8)      ;
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CERRADURA BINARIA LINEAR TAMAÑO "N"                */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA                 */
/*                   size TAMAÑO DE LA CERRADURA                    */
/*                   dir   DIRECCION                                */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN BINARIA                  */
```

```
/*-------------------------------------------------------------------------------------------------------------------*/


void im_Nbincloselin_(BIMAGEN *ime,BIMAGEN *ims,int taille,int dir)
{

        im_Nbindillin_(ime,ims,taille,dir)          ;
        im_Nbinerolin_(ims,ims,taille,(dir+4)%8)        ;
}
```

LIBRERIA : PRIMIT1.C

CONTENIDO:

1) LECTURA Y ESCRITURA DE IMAGENES BINARIAS Y NUMERICAS EN ARCHIVO.

2) ALOJAMIENTO Y LIBERACION DINAMICA DE IMAGENES BINARIAS Y NUMERICAS.

3) OPERACIONES LOGICAS ENTRE IMAGENES BINARIAS: AND, OR, COMPLEMENTACION,...

4) OPERACIONES SOBRE IMAGENES BINARIAS : ELIMINACION DE PARTICULAS TOCANDO EL BORDE DE LAS IMAGENES, ELIMINACION DE HOYOS EN LAS PARTICULAS,...

5) OPERACIONES ARITMETICAS ENTRE IMAGENES NUMERICAS: SUBSTRACCION, ADICION, INF, SUP, NEGACION, ...

6) ALGORITMOS SECUECIALES:

6.1) FUNCION DISTANCIA
6.2) RECONSTRUCION BINARIA
6.3) RECONSTRUCCION NUMERICA
6.4) ETIQUETADO DE OBJETOS

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
/*#include "f:\borlandc\ivan\logiciel\strim.h"*/
#include "c:\users\ivan\logiciel\strim.h"
/*extern int gdsmode  ;  */
int im_flag      ;
int im_bflag     ;

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: ALOJAMIENTO DINAMICO DE UNA IMAGEN NUMERICA   */
/*       ENTRADAS:   x  NUMERO DE COLUMNAS                    */
/*                   y NUMERO DE LINEAS                       */
/*                   g NUMERO DE NIVELES DE GRIS              */
/*      SALIDAS:     APUNTADOR ESTRUCTURA IMAGEN NUMERICA     */
/*-------------------------------------------------------------------------------------------------------------*/

IMAGEN *im_alloc_grey_(int x,int y, int g)
{
        IMAGEN *ftpr                ;
        int i                       ;
```

```
        im_flag++               ;

        ftpr = (IMAGEN far *)farcalloc(1,sizeof(IMAGEN))    ;
        ftpr->sx=x    ;        ftpr->sy=y    ; ftpr->lg=g  ;
        if(im_flag>NUMIMA){
                printf("\n ERROR Number Max._of Images %d ",NUMIMA) ;
                return(0)       ;
        }

        for(i=0 ; i<ftpr->sy+2 ; i++){
                ftpr->p[i] = (unsigned char far *)farcalloc(ftpr->sx+2,sizeof(char))  ;
                if(ftpr->p[i]==0){
                        printf("\n\a\a Error Allocacion im_alloc_grey");
                        exit(1)  ;
                }
        }
        return(ftpr)    ;

}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: LIBERACION DINAMICA DE UNA IMAGEN NUMERICA      */
/*       ENTRADAS:                                              */
/*                      APUNTADOR ESTRUCTURA IMAGEN NUMERICA    */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_free_grey_(IMAGEN *ftpr)
{
        int i                           ;

        im_flag--               ;

        for(i=0 ; i<(ftpr->sy+2) ; i++)
                farfree((unsigned char far *)ftpr->p[i]);
        farfree((IMAGEN far *)ftpr)     ;
}


/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: LECTURA DE ARCHIVO DE  UNA IMAGEN NUMERICA      */
/*       ENTRADAS:    *fichier ARCHIVO IMAGEN                   */
/*                    entete ENCABEZADO                         */
/*      SALIDAS:      APUNTADOR ESTRUCTURA IMAGEN NUMERICA      */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_readfileimage_(char *fichier,int entete,IMAGEN *image)
{

        int i,j         ;
        FILE *fp1       ;
        unsigned char tete[512]         ;
        unsigned char *sortie           ;
```

```
        printf("\n  READING IMAGE FILE ... %s  ",fichier)      ;

        fp1  =  fopen(fichier,"r+b")                            ;

        if(fp1 = =0)printf("\n ERRRRROOOOOR OPENING FILE %s",fichier) ;
        fread(tete,sizeof(unsigned char),entete,fp1)            ;

        for(j=0 ; j<SIZEIMY ; j++){
                sortie  =  &image->p[j+1][1]            ;
                fread(sortie,1,SIZEIMX,fp1)             ;
        }

        fclose(fp1)      ;


}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: ESCRITURA EN ARCHIVO DE  UNA IMAGEN NUMERICA    */
/*       ENTRADAS:                                              */
/*                      *tete  CONTENIDO DEL ENBABEZADO         */
/*                      entete TAMAÑO DEL ENCABEZADO            */
/*                      APUNTADOR ESTRUCTURA IMAGEN NUMERICA    */
/*      SALIDAS:            *fichier  NOMBRE DEL ARCHIVO        */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_writefileimage_(char *fichier,int entete,usc *tete, IMAGEN *image)
{

        int i,j             ;
        FILE *fp1       ;
        unsigned char *sortie           ;

        printf("\n  WRITING IMAGE FILE ... %s  ",fichier)      ;

        fp1  =  fopen(fichier,"w+b")                            ;

        if(fp1 = =0)printf("\n ERRRRROOOOOR OPENING FILE %s",fichier) ;

        fwrite(tete,sizeof(unsigned char),entete,fp1)           ;

        for(j=0 ; j<SIZEIMY ; j++){
                sortie  =  &image->p[j+1][1]            ;
                fwrite(sortie,sizeof(unsigned char),SIZEIMX,fp1)                ;
        }


        fclose(fp1)      ;


}

/*-------------------------------------------------------------------------------------------------------------------*/
```

```
/*      TRANSF: ALOJAMIENTO DINAMICO DE UNA IMAGEN BINARIA      */
/*       ENTRADAS:   x  NUMERO DE COLUMNAS                      */
/*                   y NUMERO DE LINEAS                         */
/*                   g NUMERO DE NIVELES DE GRIS                */
/*      SALIDAS:     APUNTADOR ESTRUCTURA IMAGEN BINARIA        */
/*-----------------------------------------------------------------------------------------------------------*/

BIMAGEN *im_alloc_bin_(int x,int y, int g)
{
        BIMAGEN *ftpr                   ;
        int i                           ;

        im_bflag++              ;

        ftpr = (BIMAGEN far *)farcalloc(1,sizeof(BIMAGEN)) ;
        ftpr->sx=x    ;         ftpr->sy=y    ; ftpr->lg=g  ;
        if(im_bflag>BINIMA){
                printf("\n ERROR Number Max. of Images %d ",BINIMA) ;
                return(0)       ;
        }

        for(i=0 ; i<ftpr->sy ; i++){
                ftpr->p[i] = (unsigned int far *)farcalloc((ftpr->sx/16),sizeof(unsigned int))
;
                if(ftpr->p[i]==0){
                        printf("\n\a\a Error Allocacion im_alloc_bin");
                        exit(1) ;
                }
        }
        return(ftpr)    ;

}

/*-----------------------------------------------------------------------------------------------------------*/
/*      TRANSF: LIBERACION DINAMICA DE UNA IMAGEN BINARIA       */
/*       ENTRADAS:                                              */
/*                   APUNTADOR ESTRUCTURA IMAGEN BINARIA        */
/*-----------------------------------------------------------------------------------------------------------*/

void im_free_bin_(BIMAGEN *ftpr)
{
        int i                           ;

        im_bflag--              ;

        for(i=0 ; i<ftpr->sy ; i++)
                farfree((unsigned int far *)ftpr->p[i])  ;
        farfree((BIMAGEN far *)ftpr)  ;
}

/*-----------------------------------------------------------------------------------------------------------*/
/*      TRANSF: LECTURA DE ARCHIVO DE  UNA IMAGEN BINARIA       */
/*       ENTRADAS:   *fichier ARCHIVO IMAGEN                    */
/*                   entete ENCABEZADO                          */
```

```
/*      SALIDAS:      APUNTADOR ESTRUCTURA IMAGEN BINARIA        */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_readfilebimage_(char *fichier,int entete,BIMAGEN *image)
{

        int i,j         ;
        FILE *fp1       ;
        unsigned char tete[512]         ;
        unsigned int *sortie            ;

        printf("\n  READING IMAGE FILE ... %s  ",fichier)      ;

        fp1  =  fopen(fichier,"r+b")                            ;

        if(fp1 = =0)printf("\n ERRRRROOOOOR OPENING FILE %s",fichier) ;
        fread(tete,sizeof(unsigned char),entete,fp1)            ;

        for(j=0 ; j<SIZEIMY ; j++){
                sortie  =  &image->p[j][0]              ;
                fread(sortie,sizeof(unsigned int),SIZEIMX/16,fp1)               ;
        }


        fclose(fp1)     ;


}


/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: ESCRITURA EN ARCHIVO DE  UNA IMAGEN BINARIA       */
/*       ENTRADAS:                                                */
/*                      *tete  CONTENIDO DEL ENBABEZADO           */
/*                      entete TAMAÑO DEL ENCABEZADO              */
/*                      APUNTADOR ESTRUCTURA IMAGEN BINARIA       */
/*      SALIDAS:            *fichier  NOMBRE DEL ARCHIVO          */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_writefilebimage_(char *fichier,int entete,usc *tete, BIMAGEN *image)
{

        int i,j         ;
        FILE *fp1       ;
        unsigned int *sortie            ;

        printf("\n  WRITING IMAGE FILE ... %s  ",fichier)      ;

        fp1  =  fopen(fichier,"w+b")                            ;

        if(fp1 = =0)printf("\n ERRRRROOOOOR OPENING FILE %s",fichier) ;

        fwrite(tete,sizeof(unsigned char),entete,fp1)           ;
```

```
        for(j=0 ; j<SIZEIMY ; j++){
                sortie = &image->p[j][0]                ;
                fwrite(sortie,sizeof(unsigned int),SIZEIMX/16,fp1)              ;
        }


        fclose(fp1)     ;


}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: LECTURA DE ENCABEZADO IMAGEN NUMERICA O BINARIA      */
/*       ENTRADAS:   *fichier ARCHIVO IMAGEN                         */
/*                   entete TAMAÑO ENCABEZADO                        */
/*      SALIDAS:     *tete CONTENIDO DEL ENCABEZADO                  */
/*-------------------------------------------------------------------------------------------------------------*/

void im_readheadimage_(char *fichier,int entete, usc *tete)
{

        FILE *fp1       ;

        printf("\n  READING HEAD IMAGE FILE ... %s ",fichier)       ;

        fp1 = fopen(fichier,"r+b")                      ;

        if(fp1==0)printf("\n ERRRRROOOOOR OPENING FILE %s",fichier) ;
        fread(tete,sizeof(unsigned char),entete,fp1)            ;

        fclose(fp1)     ;


}


/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: MULTIPLICACION DE UNA IMAGEN POR UNA CONSTANTE*/
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA                 */
/*                   val  CONSTANTE                                  */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                  */
/*-------------------------------------------------------------------------------------------------------------*/

void im_greycmul_(IMAGEN *ime, IMAGEN *ims,unsigned char val)
{
        int i,j                 ;

        for(i=1; i<=ime->sy ;i++)
                for(j=1 ; j<=ime->sx ; j++)
                        ims->p[i][j] = val * ime->p[i][j]       ;
}


/*-------------------------------------------------------------------------------------------------------------*/
```

```
/*      TRANSF: NEGACION DE UNA IMAGEN NUMERICA                         */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA                    */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN NUMERICA                    */
/*--------------------------------------------------------------------------------------------------------------------*/

void im_greyinv_(IMAGEN *ime, IMAGEN *ims)
{
        int i,j                         ;

        for(i=1; i<=ime->sy ;i++)
                for(j=1 ; j<=ime->sx ; j++)
                        ims->p[i][j] = (ime->lg-1) - ime->p[i][j]       ;


}


/*--------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  BORRADO DE UNA IMAGEN NUMERICA                         */
/*       ENTRADAS:   image  ESTRUCTURA IMAGEN NUMERICA                  */
/*      SALIDAS:      image ESTRUCTURA IMAGEN NUMERICA                  */
/*--------------------------------------------------------------------------------------------------------------------*/

void im_clear_(IMAGEN *image)
{
        int i,j                                 ;


        for(i=1 ; i<=image->sy ; i++)
                for(j=1 ; j<=image->sx ; j++)
                        image->p[i][j] = 0      ;

}
/*--------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  BORRADO DE UNA IMAGEN BINARIA                          */
/*       ENTRADAS:   image  ESTRUCTURA IMAGEN BINARIA                   */
/*      SALIDAS:      image ESTRUCTURA IMAGEN BINARIA                   */
/*--------------------------------------------------------------------------------------------------------------------*/

void im_bclear_(BIMAGEN *ime)
{
        int i,j                                 ;


        for(i=0 ; i<ime->sy ; i++)
                for(j=0 ; j<(ime->sx/16) ; j++)
                        ime->p[i][j] = 0        ;

}

/*--------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  CALCULO DEL VOLUMEN DE UNA IMAGEN                      */
```

```
/*      ENTRADAS:   image  ESTRUCTURA IMAGEN NUMERICA          */
/*      SALIDAS:       VOLUMEN DE LA IMAGEN                    */
/*--------------------------------------------------------------------------------------------------------------*/

long im_volumen_(IMAGEN *image)
{
        unsigned long  vol,buf_his[512]        ;
        int i,j                                ;


        im_fhisto_(image,buf_his)                      ;
        vol = 0                                ;
        for(i=0 ; i<image->lg ; i++)
                vol += buf_his[i]*i                                    ;


        return((long)vol)        ;
}


/*--------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  CALCULO DEL VOLUMEN DE UNA IMAGEN              */
/*      ENTRADAS:   image  ESTRUCTURA IMAGEN NUMERICA           */
/*      SALIDAS:       *buf HISTOGRAMA DE LA IMAGEN             */
/*--------------------------------------------------------------------------------------------------------------*/

void im_fhisto_(IMAGEN *image,unsigned long *buf)
{
        int i,j             ;
        for(i=0 ; i<image->lg ; i++)buf[i] = 0       ;

        for(i=1 ; i<=image->sy  ; i++)
                for(j=1 ; j<=image->sx; j++)
                        buf[(int)image->p[i][j]]++      ;

}

/*--------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: SUBSTRACCION DE UNA IMAGEN POR UNA CONSTANTE*/
/*      ENTRADAS:   image  ESTRUCTURA IMAGEN NUMERICA           */
/*                  val   CONSTANTE                             */
/*      SALIDAS:       image ESTRUCTURA IMAGEN NUMERICA         */
/*--------------------------------------------------------------------------------------------------------------*/

void im_greysub_const_(IMAGEN *image,unsigned char val)
{

        int i,j                          ;

        for(i=1; i<=image->sy ;i++)
                for(j=1 ; j<=image->sx ; j++)
                        if(val>image->p[i][j])image->p[i][j]=0          ;
                        else image->p[i][j] -= val       ;
```

```
}


unsigned char val_min_min_(usc a,usc b,usc c,usc d,usc e)
{

        unsigned char minv    ;
        minv = a                        ;

        if(minv > b) minv = b ;
        if(minv > c) minv = c ;
        if(minv > d) minv = d ;
        minv + = 1                              ;
        if(minv > e)minv = e  ;

        return(minv)    ;
}


unsigned char val_min_4_(usc a,usc b,usc c,usc d)
{

        unsigned char minv    ;
        minv = a                        ;

        if(minv > b) minv = b ;
        if(minv > c) minv = c ;
        if(minv > d) minv = d ;

        return(minv)    ;
}


int readpix(int x1,int y1, IMAGEN *image)
{

        int nouv        ;


        nouv = (int)image->p[y1][x1] ;

        return(nouv)            ;

}




int im_rowr_(int y1,IMAGEN *image, unsigned char *buf)
{

        int     i ;

        for(i=0 ; i<image->sx ; i++)
```

```
                buf[i] = (unsigned char)image->p[y1][i]        ;

return(0)       ;
}


int im_roww_(int y1,IMAGEN *image, unsigned char *buf)
{

        int     i ;

        for(i=0 ; i<image->sx ; i++)
                image->p[y1][i] = (char)buf[i] ;

return(0)       ;
}


int im_columr_(int x1,IMAGEN *image, unsigned char *buf)
{

        int     i ;

        for(i=0 ; i<image->sy ; i++)
                buf[i] = (unsigned char)image->p[i][x1]        ;

return(0)       ;
}


int im_columw_(int x1,IMAGEN *image, unsigned char *buf)
{

        int     i ;

        for(i=0 ; i<image->sy ; i++)
                image->p[i][x1] = (char)buf[i] ;

return(0)       ;
}


/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CALCULO DE LA FUNCION DISTANCIA                                 */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN BINARIA                             */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN NUMERICA                          */
/*-------------------------------------------------------------------------------------------------------------*/

void im_distance_(BIMAGEN *ime, IMAGEN *ims)
{
```

```
        int i,j,k                       ;
        unsigned int vmask                      ;

/**********  Inicialisazion de buf_im        ***************/

        for(i=1; i<=ime->sy ;i++)
                for(j=0 ; j<(ime->sx/16) ; j++){
                        vmask = 0x8000                  ;
                        for(k=1 ; k<=16 ; k++){
                        if((ime->p[i-1][j] & vmask)==0)
                                ims->p[i][j*16+k]= BLACK   ;
                        else ims->p[i][j*16+k] = WHITE          ;
                        vmask = vmask >> 1          ;
                        }
                }

        for(i=0 ; i<(ims->sx+2) ;i++){
                ims->p[0][i]= BLACK             ;
                ims->p[ims->sy+1][i]=BLACK      ;
                ims->p[i][0]=BLACK              ;
                ims->p[i][ims->sx+1]=BLACK      ;
        }

/*****************  DISTANCE (PREMIERE PASSAGE)        ***************/

for(i=1; i<=ime->sy ;i++)
        for(j=1 ; j<=ime->sx ; j++)
                ims->p[i][j] = val_min_min_(ims->p[i-1][j-1],ims->p[i-1][j],
                                        ims->p[i-1][j+1],ims->p[i][j-1],ims->p[i][j])
;
for(i=ime->sy; i>0 ;i--)
        for(j=ime->sx ; j>0 ; j--)
                ims->p[i][j] = val_min_min_(ims->p[i+1][j+1],ims->p[i+1][j],
                                        ims->p[i+1][j-1],ims->p[i][j+1],ims->p[i][j])
;


}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: ETIQUETADO DE OBJETOS                                                  */
/*       ENTRADAS:   ima  ESTRUCTURA IMAGEN BINARIA                                    */
/*      SALIDAS:      imsn ESTRUCTURA IMAGEN NUMERICA                                 */
/*-------------------------------------------------------------------------------------------------------------*/

void im_labelobjet_(BIMAGEN *ima, IMAGEN *imsn)
{
        int x,y,i,j,ngr;
        usc val ;
```

```
        BIMAGEN *ime,*ims ;

        ime = im_alloc_bin_(ima->sx,ima->sy,ima->lg) ;
        ims = im_alloc_bin_(ima->sx,ima->sy,ima->lg) ;

im_bcopy_(ima,ime) ;
im_clear_(imsn) ;
im_bclear_(ims) ;
ngr=-1;
while(im_area_(ime)!=0){
ngr++;
y= -1 ;
do{
        y++ ;
        x=-1   ;
        do{
        x++;
        val =(usc)((ime->p[y][x/16] & (0x8000 >> (x % 16))) != 0 ? 1 : 0)        ;
        }while((usc)val == 0 && x<ime->sx) ;
}while((usc)val == 0 && y<ime->sy) ;

im_bwritepix_(ims,x,y,1) ;
im_fastbuild_(ime,ims) ;
im_greysetmask0_(imsn,ims,imsn,(usc)(ngr%255)+1) ;
for(i=0; i<ime->sy ;i++)
        for(j=0 ; j<(ime->sx/16) ; j++)
                ime->p[i][j] = (ims->p[i][j] ^ 0xFFFF) & ime->p[i][j]  ;

}

}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: RECOSTRUCCION PRIMER OBJETO EN LA IMAGEN          */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  BINARIA              */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA               */
/*-------------------------------------------------------------------------------------------------------------*/

void im_firstobjetbuild_(BIMAGEN *ime,BIMAGEN *ims)
{
        int x,y  ;
        usc val ;

        im_bclear_(ims) ;
y= -1 ;
do{
        y++ ;
        x=-1   ;
        do{
        x++;
        val =(usc)((ime->p[y][x/16] & (0x8000 >> (x % 16))) != 0 ? 1 : 0)        ;
        }while((usc)val == 0 && x<ime->sx) ;
}while((usc)val == 0 && y<ime->sy) ;
```

```
im_bwritepix_(ims,x,y,1) ;
im_fastbuild_(ime,ims) ;


}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: RECOSTRUCCION BINARIA DE UNA IMAGEN A PARTIR DE UN
MARCADOR   */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  BINARIA                    */
/*       SALIDAS:      mark ESTRUCTURA IMAGEN BINARIA                   */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_fastbuild_(BIMAGEN *ime, BIMAGEN *mark)
{

        int i,j,k          ;
        unsigned int vmask,val;
        long drap, drap1        ;



        im_binand_(ime,mark)  ;
        drap=1          ;

while(drap!=0){
        drap  =  0      ;
        drap1  =  1     ;
        while(drap1 !=  0){
                drap1  =  0 ; vmask  =  0 ;
                for(j=0 ; j<(ime->sx/16) ; j++){
                val  =  mark->p[0][j]    ;
                mark->p[0][j] |= (((val >> 1) | vmask) & ime->p[0][j])   ;
                vmask  =  val << 15   ; drap1  +=  (val ^ mark->p[0][j]) ;
                drap  +=  drap1 ;
                }
        }
        drap1  =  1     ;
        while(drap1 !=0){
                for(i=1 ; i<ime->sy ; i++){
                        for(j=0 ; j<(ime->sx/16) ; j++)
                        mark->p[i][j] |= (mark->p[i-1][j] & ime->p[i][j]) ;
                        drap1  =  0 ; vmask  =  0 ;
                        for(j=0 ; j<(ime->sx/16) ; j++){
                        val  =  mark->p[i][j]    ;
                        mark->p[i][j] |= (((val >> 1) | vmask) & ime->p[i][j]) ;
                        vmask  =  val << 15   ; drap1  +=  (val ^ mark->p[i][j]) ;
                        drap  +=  drap1 ;
                        }
                }
        }

        drap1  =  1     ;
```

```
        while(drap1 != 0){
                drap1 = 0 ; vmask = 0 ;
                for(j=(ime->sx/16)-1 ; j>=0 ; j--){
                val = mark->p[255][j] ;
                mark->p[255][j] |= (((val << 1) | vmask) & ime->p[255][j])  ;
                vmask = val >> 15  ; drap1 += (val ^ mark->p[255][j]) ;
                drap += drap1 ;
                }
        }
        drap1 = 1      ;
        while(drap1 !=0){
                for(i=ime->sy-2 ; i>=0 ; i--){
                        for(j=(ime->sx/16)-1 ; j>=0 ; j--)
                        mark->p[i][j] |= (mark->p[i+1][j] & ime->p[i][j]) ;

                        drap1 = 0 ; vmask = 0 ;
                        for(j=(ime->sx/16)-1 ; j>=0 ; j--){
                        val = mark->p[i][j]     ;
                        mark->p[i][j] |= (((val << 1) | vmask) & ime->p[i][j]) ;
                        vmask = val >> 15  ; drap1 += (val ^ mark->p[i][j]) ;
                        drap += drap1 ;
                        }
                }
        }


}

}


/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: RECOSTRUCCION NUMERICA DE UNA IMAGEN A PARTIR DE UN
MARCADOR   */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  NUMERICA              */
/*      SALIDAS:      mark ESTRUCTURA IMAGEN NUMERICA                 */
/*-------------------------------------------------------------------------------------------------------------*/

void im_greybuild_(IMAGEN *ime, IMAGEN *mark)
{

        int i,j,k          ;
        usc val                 ;
        long drap               ;



for(i=0 ; i<ime->sy ; i++)ime->p[i][0]=0 ;
for(i=0 ; i<ime->sy ; i++)ime->p[i][ime->sy+1]=0 ;
for(i=0 ; i<ime->sx ; i++)ime->p[0][i]=0 ;
for(i=0 ; i<ime->sx ; i++)ime->p[ime->sx+1][i]=0 ;
for(i=0 ; i<ime->sy ; i++)mark->p[i][0]=0 ;
for(i=0 ; i<ime->sy ; i++)mark->p[i][mark->sy+1]=0 ;
for(i=0 ; i<ime->sx ; i++)mark->p[0][i]=0 ;
```

```
for(i=0 ; i<ime->sx ; i++)mark->p[mark->sx+1][i]=0 ;

drap=1 ;
while(drap!=0){
drap = 0        ;

for(i=1; i<=ime->sy ;i++){
        for(j=1 ; j<=ime->sx ; j++){
                val = mark->p[i][j]             ;
                mark->p[i][j]=val_min_max_(mark->p[i-1][j-1],mark->p[i-1][j],mark->p[i-
1][j+1],
                        mark->p[i][j-1],mark->p[i][j],ime->p[i][j]) ;
                drap += (val ^ mark->p[i][j]) ;
                }
}


for(i=(ime->sy); i>=1 ;i--){
        for(j=(ime->sx) ; j>=1 ; j--){
                val = mark->p[i][j]             ;
                mark->p[i][j] = val_min_max_(mark->p[i+1][j+1],mark->p[i+1][j],mark-
>p[i+1][j-1],
                        mark->p[i][j+1],mark->p[i][j],ime->p[i][j])     ;
                drap += (val ^ mark->p[i][j]) ;
        }

}
/*printf("\r %ld   ",drap)        ; */

}

}

unsigned char val_min_max_(usc am,usc bm,usc cm,usc dm,usc em, usc e)
{

        unsigned char minv     ;
        minv = am                       ;

        if((usc)minv < (usc)bm) minv = bm   ;
        if((usc)minv < (usc)cm) minv = cm   ;
        if((usc)minv < (usc)dm) minv = dm   ;
        if((usc)minv < (usc)em) minv = em   ;

        if((usc)minv > (usc)e)minv = e          ;

        return(minv)    ;
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: COPIADO DE UNA  PARTE DE LA IMAGEN (USANDO MARCADOR)*/
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  NUMERICA               */
/*                   mark ESTRUCTURA IMAGEN BINARIA                 */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                 */
```

```
/*-----------------------------------------------------------------------------------------------------------------*/

void im_greysetmask0_(IMAGEN *ime,BIMAGEN *mask, IMAGEN *ims, unsigned char val)
{

        int i,j,k                       ;
        unsigned int vmask                      ;


        for(i=1; i<=ime->sy ;i++)
                for(j=0 ; j<(ime->sx/16) ; j++){
                        vmask = 0x8000                  ;
                        for(k=1 ; k<=16 ; k++){
                        if((mask->p[i-1][j] & vmask)==0)
                                ims->p[i][j*16+k]=ime->p[i][j*16+k]     ;
                        else ims->p[i][j*16+k] = val                    ;
                        vmask = vmask >> 1      ;
                        }
                }


}

/*-----------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: SUP ENTRE DOS IMAGENES NUMERICAS                                */
/*       ENTRADAS:   ime1  ESTRUCTURA IMAGEN  NUMERICA                          */
/*                   ime2 ESTRUCTURA IMAGEN NUMERICA                            */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                             */
/*-----------------------------------------------------------------------------------------------------------------*/

void im_greysup_(IMAGEN *ime1, IMAGEN *ime2, IMAGEN *ims)
{

        int i,j                                 ;

        for(i=1; i<=ime1->sy ;i++)
                for(j=1 ; j<=ime1->sx ; j++)
                        if(ime1->p[i][j]>ime2->p[i][j])ims->p[i][j]=ime1->p[i][j]       ;
                        else ims->p[i][j]= ime2->p[i][j]                                ;
}

/*-----------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: INF ENTRE DOS IMAGENES NUMERICAS                                */
/*       ENTRADAS:   ime1  ESTRUCTURA IMAGEN  NUMERICA                          */
/*                   ime2 ESTRUCTURA IMAGEN NUMERICA                            */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                             */
/*-----------------------------------------------------------------------------------------------------------------*/

void im_greyinf_(IMAGEN *ime1, IMAGEN *ime2, IMAGEN *ims)
{

        int i,j                                 ;

        for(i=1; i<=ime1->sy ;i++)
```

```
                for(j=1 ; j<=ime1->sx ; j++)
                        if(ime1->p[i][j]>ime2->p[i][j])ims->p[i][j]=ime2->p[i][j]       ;
                        else ims->p[i][j]= ime1->p[i][j]                                ;


}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: ELIMINACION DE OBJETOS TOCANDO LOS BORDES                            */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  BINARIA                                 */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA                                  */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_binedgeoff_(BIMAGEN *ime, BIMAGEN *ims)
{
        int i,j   ;

        im_bclear_(ims) ;
        for(j=0 ; j<(ime->sx/16) ; j++)
                ims->p[0][j] = 0xFFFF;
        for(j=0 ; j<(ime->sx/16) ; j++)
                ims->p[ims->sy-1][j] = 0xFFFF       ;
        for(i=0 ; i<ime->sy ;i++){
                        ims->p[i][0] = 0x8000       ;
                        ims->p[i][ims->sx-1] = 0x0001       ;
        }

        im_fastbuild_(ime,ims) ;
        im_bindif_(ime,ims)   ;
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: ELIMINACION DE HOYOS EN LOS OBJETOS                                  */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  BINARIA                                 */
/*      SALIDAS:      ims ESTRUCTURA IMAGEN BINARIA                                  */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_binfillhole_(BIMAGEN *ime,BIMAGEN *ims)
{
        int i,j   ;

        im_bclear_(ims) ;
        for(j=0 ; j<(ime->sx/16) ; j++)
                ims->p[0][j] = 0xFFFF;
        for(j=0 ; j<(ime->sx/16) ; j++)
                ims->p[ims->sy-1][j] = 0xFFFF       ;
        for(i=0 ; i<ime->sy ;i++){
                        ims->p[i][0] = 0x8000       ;
                        ims->p[i][ims->sx-1] = 0x0001       ;
        }

        im_bininv_(ime,ime)   ;
        im_fastbuild_(ime,ims) ;
        im_bindif_(ime,ims)   ;
```

```
        im_bininv_(ime,ime)     ;
        im_binor_(ime,ims)      ;
}


/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  NOT LOGICO                                                                    */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  BINARIA                                           */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN BINARIA                                          */
/*-------------------------------------------------------------------------------------------------------------*/

void im_bininv_(BIMAGEN *ime, BIMAGEN *ims)
{

        int i,j                         ;

        for(i=0 ; i<ime->sy ;i++)
                for(j=0 ; j<(ime->sx/16) ; j++)
                        ims->p[i][j] = ime->p[i][j] ^ 0xFFFF   ;

}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  AND LOGICO                                                                    */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  BINARIA                                           */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN BINARIA                                          */
/*-------------------------------------------------------------------------------------------------------------*/

void im_binand_(BIMAGEN *ime,  BIMAGEN *ims)
{

        int i,j                                 ;


        for(i=0; i<ime->sy ;i++)
                for(j=0 ; j<(ime->sx/16) ; j++)
                        ims->p[i][j] &= ime->p[i][j]    ;

}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  OR  LOGICO                                                                    */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN  BINARIA                                           */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN BINARIA                                          */
/*-------------------------------------------------------------------------------------------------------------*/

void im_binor_(BIMAGEN *ime,BIMAGEN *ims)
{

        int i,j                                 ;

        for(i=0; i<ime->sy ;i++)
                for(j=0 ; j<(ime->sx/16) ; j++)
                        ims->p[i][j] |= ime->p[i][j]    ;
```

```
}
/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  XOR LOGICO                                                                                    */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN  BINARIA                                                          */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN BINARIA                                                          */
/*-------------------------------------------------------------------------------------------------------------*/

void im_binxor_(BIMAGEN *ime,BIMAGEN *ims)
{
        int i,j             ;
        for(i=0 ; i<ime->sy; i++)
                for(j=0 ; j<(ime->sx/16) ; j++)
                        ims->p[i][j] ^=ime->p[i][j]     ;

}


/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  DIFERENCIA LOGICA                                                                             */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN  BINARIA                                                          */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN BINARIA                                                          */
/*-------------------------------------------------------------------------------------------------------------*/

void im_bindif_(BIMAGEN *ime,BIMAGEN *ims)
{

        int i,j                                 ;

        for(i=0; i<ime->sy ;i++)
                for(j=0 ; j<(ime->sx/16) ; j++)
                        ims->p[i][j] = (ims->p[i][j] ^ 0xFFFF) & ime->p[i][j]  ;


}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  COPIADO DE IMAGENES BINARIAS                                                                  */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN  BINARIA                                                          */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN BINARIA                                                          */
/*-------------------------------------------------------------------------------------------------------------*/


void im_bcopy_(BIMAGEN *ime,BIMAGEN *ims)
{

        int i,j                                 ;

        for(i=0; i<ime->sy ;i++)
                for(j=0 ; j<(ime->sx/16) ; j++)
                        ims->p[i][j] = ime->p[i][j]     ;
```

```
}

usc im_breadpix_(BIMAGEN *image,int x, int y)
{

        return((image->p[y][x/16] & (0x8000 >> (x % 16))) != 0 ? 1 : 0 ) ;
}

void im_bwritepix_(BIMAGEN *image,int x, int y, usc val)
{
        if((usc)val ==0)
        image->p[y][x/16] &= (0x7FFF >> (x % 16))       ;
        else
        image->p[y][x/16] |= (0x8000 >> (x % 16))       ;

}

long im_area_(BIMAGEN *image)
{
        int i,j,k,val     ;
        long imarea=0 ;

        for(i=0; i<image->sy ;i++)
                for(j=0 ; j<(image->sx/16) ; j++){
                        val = image->p[i][j]            ;
                        imarea += (0x0001 & val)    ;
                        for(k=0 ; k<15 ; k++){
                        val >>=1     ;
                        imarea += (val & 0x0001)    ;
                        }
                }
        return(imarea)  ;

}
void im_bshift_(BIMAGEN *ime, BIMAGEN *ims, int dir, int nite)
{

        switch(dir){
                case 0 : im_bshift_0_(ime, ims, nite)   ; break ;
                case 1 : im_bshift_1_(ime, ims, nite)   ; break ;
                case 2 : im_bshift_2_(ime, ims, nite)   ; break ;
                case 3 : im_bshift_3_(ime, ims, nite)   ; break ;
                defauflt : printf("\n NON VALABLE DIRECTION ")      ;
        }
}

void im_bshift_3_(BIMAGEN *ime, BIMAGEN *ims, int nite)
{
        int i,j,k         ;

        for(k=0 ; k<nite ; k++){
                for(i=(ime->sy-1) ; i>0 ; i--){
                        for(j=0 ; j<(ime->sx/16) ; j++)
                                ims->p[i][j] = ims->p[i-1][j]   ;
```

```
                }
                for(j=0 ; j<(ime->sx/16) ; j++)
                        ims->p[0][j] = 0        ;
        }
}

void im_bshift_2_(BIMAGEN *ime, BIMAGEN *ims, int nite)
{
        int i,j,k           ;
        unsigned int vmask0,vmask1   ;

for(k=0  ; k<nite ; k++){
        for(i=0  ; i<ime->sy  ; i++){
                vmask0 = 0x0000          ;
                for(j=(ime->sx/16)-1 ; j>=0  ; j-- ){
                vmask1 =((0x8000 & ims->p[i][j])==0x8000) ? 1 : 0 ;
                ims->p[i][j] = (ims->p[i][j] << 1 ) | vmask0 ;
                vmask0=vmask1        ;
                }
        }
}
}

void im_bshift_1_(BIMAGEN *ime, BIMAGEN *ims, int nite)
{
        int i,j,k           ;

        for(k=0 ; k<nite ; k++){
                for(i=0 ; i<(ime->sy-1) ; i++){
                        for(j=0 ; j<(ime->sx/16) ; j++)
                                ims->p[i][j] = ims->p[i+1][j] ;
                }

                for(j=0 ; j<(ime->sx/16) ; j++)
                        ims->p[ime->sy-1][j] = 0      ;
        }
}

void im_bshift_0_(BIMAGEN *ime, BIMAGEN *ims, int nite)
{
        int i,j,k           ;
        unsigned int vmask0,vmask1   ;

for(k=0  ; k<nite ; k++){
        for(i=0  ; i<ime->sy  ; i++){
                vmask0 = 0x0000          ;
                for(j=0 ; j<(ime->sx/16) ; j++ ){
                vmask1 = 0x0001 & ims->p[i][j]      ;
                vmask1 =((0x0001 & ims->p[i][j])==0x0001) ? 0x8000 : 0 ;
                ims->p[i][j] = (ims->p[i][j] >> 1 ) | vmask0 ;
                vmask0=vmask1        ;
                }
        }
}
```

```
}


unsigned char val_set_(usc a,usc b,usc c,usc d,usc e)
{

        unsigned char vals      ;

        vals = (a & e) | (b & e) | (c & e) | (d & e)     ;
        if(vals = = WHITE)return(WHITE)        ;
        return(BLACK)           ;

}


/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF:  COPIADO DE IMAGENES NUMERICAS                                  */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN  NUMERICA                          */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN  NUMERICA                         */
/*-------------------------------------------------------------------------------------------------------------*/

void im_copy_(IMAGEN *ime, IMAGEN *ims)
{

        int i,j                         ;

        for(i=1 ; i<=ime->sy ;i++)
                for(j=1 ; j<=ime->sx ; j++)
                        ims->p[i][j] = ime->p[i][j]      ;

}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: SUBSTRACCION ENTRE DOS IMAGENES NUMERICAS       */
/*       ENTRADAS:    ime1  ESTRUCTURA IMAGEN NUMERICA          */
/*                    ime2  ESTRUCTURA IMAGEN NUMERICA          */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN NUMERICA          */
/*-------------------------------------------------------------------------------------------------------------*/

void im_greysub_(IMAGEN *ime1, IMAGEN *ime2, IMAGEN *ims)
{

        int i,j             ;

        for(i=1; i<=ime1->sy ;i++)
                for(j=1 ; j<=ime1->sx ; j++){
/*                      if((ime1->p[i][j]-ime2->p[i][j])>0)  */
                        ims->p[i][j] = ime1->p[i][j] - ime2->p[i][j]     ;
/*                      else
                        ims->p[i][j]=0 ;  */
                }
```

```
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: ADICION ENTRE DOS IMAGENES NUMERICAS                                                           */
/*       ENTRADAS:   ime1  ESTRUCTURA IMAGEN NUMERICA                                                          */
/*                   ime2  ESTRUCTURA IMAGEN NUMERICA                                                          */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                                                            */
/*-------------------------------------------------------------------------------------------------------------*/

void im_greyadd_(IMAGEN *ime1, IMAGEN *ime2, IMAGEN *ims)
{

        int i,j             ;

        for(i=1; i<=ime1->sy ;i++)
                for(j=1 ; j<=ime1->sx ; j++){
                        ims->p[i][j] = ime1->p[i][j] + ime2->p[i][j]   ;
                }


}
```

LIBRERIA : PRIMIT1.C

CONTENIDO:

1) TRANSFORMACIONES MAS COMPLEJAS.

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include "c:\users\ivan\logiciel\strim.h"

extern int gdsmode     ;
extern int im_flag     ;

/*************************************************************************/
/*      im_skeleton_(ime,ims,size)                                       */
/*      ime : INPUT IMAGE (Binary Image)                                 */
/*      ims : OUTPUT IMAGE (Binary Image)                                */
/*      size : "vitesse parameter"                                       */
/*              size = = 1  Morphological Skeleton                           */
/*              size > 1 Conditional Bisector                            */
/*************************************************************************/



void im_skeleton_(BIMAGEN *ime, BIMAGEN *ims, int size)
{

        IMAGEN *nw1,*nw2                ;


        nw1 = im_alloc_grey_(ime->sx,ime->sy,LEVELG)    ;
        nw2 = im_alloc_grey_(ime->sx,ime->sy,LEVELG)    ;

        im_distance_(ime,nw1)           ;


        im_Ngreydilate_(nw1,nw2,size)                   ;

        im_greysub_(nw2,nw1,nw1)                        ;

        im_thresh_(nw1,ims,1,255)                       ;

        im_bininv_(ims,ims)             ;

        im_binand_(ime,ims)                     ;
```

```
        im_free_grey_(nw1)          ;
        im_free_grey_(nw2)          ;

}

/*****************************************************************/
/*      im_quench_(ime,ims,size)                                 */
/*      ime : INPUT IMAGE (Binary Image)                         */
/*      ims : OUTPUT IMAGE (Binary Image)                        */
/*      size : "vitesse parameter"                               */
/*              size==1  Morphological Skeleton                      */
/*              size>1 Conditional Bisector                      */
/*****************************************************************/


void im_quench_(BIMAGEN *ime, IMAGEN *ims, int size)
{

        IMAGEN *nw2             ;
        BIMAGEN *bw1        ;


        nw2 = im_alloc_grey_(ims->sx,ims->sy,ims->lg)    ;

        bw1 = im_alloc_bin_(ime->sx,ime->sy,ime->lg)     ;

        im_distance_(ime,ims)               ;


        im_Ngreydilate_(ims,nw2,size)           ;

        im_greysub_(nw2,ims,nw2)                        ;

        im_thresh_(nw2,bw1,1,255)                       ;

        im_bininv_(bw1,bw1)             ;

        im_binand_(ime,bw1)                     ;

        im_bininv_(bw1,bw1)         ;

        im_greysetmask0_(ims,bw1,ims,0)     ;


        im_free_bin_(bw1)           ;
        im_free_grey_(nw2)          ;

}

/*---------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: RECOSNTRUCCION A PARTIR DE FUNCION QUENCH       */
/*       ENTRADAS:    ime  ESTRUCTURA IMAGEN NUMERICA            */
/*      SALIDAS:       ims ESTRUCTURA IMAGEN BINARIA             */
```

```
/*-------------------------------------------------------------------------------------------------------------------*/

void im_buildquench_(IMAGEN *ime, BIMAGEN *ims)
{

        usc max                 ;
        int i,j         ;
        BIMAGEN *bw1        ;

        bw1  = im_alloc_bin_(ims->sx,ims->sy,ims->lg)       ;
        im_bclear_(ims);
        max = 0                 ;
        for(i=1 ; i<=ime->sy ; i++)
                for(j=0 ; j<=ime->sx ; j++)
                        if((usc)max<(usc)ime->p[i][j])max=ime->p[i][j] ;
        im_thresh_(ime,ims,max,255)  ;

        for(i=((int)max-1) ; i>0 ; i--){
                im_bindilate_(ims)              ;
                im_thresh_(ime,bw1,(usc)i,255)          ;
                im_binor_(bw1,ims)              ;
        }


        im_free_bin_(bw1)       ;

}


/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: GRANULOMETRIA POR APERTURAS                                         */
/*       ENTRADAS:   bima  ESTRUCTURA IMAGEN BINARIA                                */
/*                   step   CONSTANTE                                               */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                                 */
/*                   *vecgra FUNCION DE DISTIBUKCION GRANULOMET.  */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_sizedist_bin_(BIMAGEN *bima,IMAGEN *ime, float *vecgra, int step)
{

        int i,j   ;
        usc max                 ;
        BIMAGEN *bime       ;


        bime  = im_alloc_bin_(bima->sx,bima->sy,bima->lg)           ;


        im_distance_(bima,ime)          ;
        for(i=1 ; i<(ime->sy-1) ; i++)
                for(j=1 ; j<(ime->sy-1) ; j++)
                        if((usc)max<(usc)ime->p[i][j])max=ime->p[i][j] ;

        i=0 ;
```

```
        vecgra[0] = (float)im_area_(bima)              ;


        while(i<=(int)max){
                i+= step       ;
                im_thresh_(ime,bime,i,LEVELG-1)              ;
                vecgra[i] = vecgra[0]-(float)im_area_(bime)  ;
                im_greydilate_(ime)                          ;
        }

        for(j=i ; j<=255 ; j++)vecgra[j] =vecgra[0] ;
        vecgra[0]= 0.0 ;

        im_free_bin_(bime)      ;

}

void im_cova_bin_(BIMAGEN *bima,float *vecgra,int step,int maxs,int dir)
{

        int i,j   ;
        usc max                 ;
        BIMAGEN *bime1,*bime2,*bime3     ;

        bime1 = im_alloc_bin_(bima->sx,bima->sy,bima->lg)            ;
        bime2 = im_alloc_bin_(bima->sx,bima->sy,bima->lg)            ;
        bime3 = im_alloc_bin_(bima->sx,bima->sy,bima->lg)            ;
        im_bcopy_(bima,bime1)          ;
        im_bclear_(bime3)       ; im_bininv_(bime3,bime3) ;
        for(i=0 ; i<maxs ; i++)vecgra[i]=0.0         ;
        for(i=1,j=0; i<maxs ; i +=step,j++){
                printf("\r  %d  ",i)       ;
             im_bshift_(bime1,bime1,dir,step) ;
             im_bshift_(bime3,bime3,dir,step) ;
             im_bcopy_(bime1,bime2)          ;
             im_binand_(bima,bime2)          ;
             vecgra[j]=(float)im_area_(bime2)/(float)im_area_(bime3) ;
             }

        im_free_bin_(bime1)     ;
        im_free_bin_(bime2)     ;
        im_free_bin_(bime3)     ;

}
```

/**/

```
/* TRANSF:  GRADIENTE MORFOLOGICO                          */
/*       im_Ngradient_(ime,ims,size,type)                  */
/*       ime : INPUT IMAGE (Binary Image)                  */
/*       ims : OUTPUT IMAGE (Binary Image)                 */
/*       size : SIZE OF THE STRUCTURE                              */
/*     type : 2  DILATION - ORIGINAL IMAGE                 */
/*       --     1  DILATION - EROSION                      */
/*              0  ORIGINAL IMAGE - EROSION                */
/*************************************************************/

void im_Ngradient_(IMAGEN *ime,IMAGEN *ims,int size, int type)
{

        IMAGEN *nw1 ;

        switch(type){
        case 2 :
                im_Ngreydilate_(ime,ims,size)  ;
                im_greysub_(ime,ims,ims)       ;
                break ;
        case 1 :
                nw1  = im_alloc_grey_(ime->sx,ime->sy,ime->lg) ;
                im_Ngreydilate_(ime,nw1,size) ;
                im_Ngreyerode_(ime,ims,size)  ;
                im_greysub_(nw1,ims,ims)      ;
                im_free_grey_(nw1)            ;
                break ;
        case 0 :
                im_Ngreyerode_(ime,ims,size)  ;
                im_greysub_(ims,ime,ims)              ;
                break ;
        default : printf("\n * type of Gradient* only 0 1 or 2 ")        ;
        }
}

/*************************************************************/
/* TRANSF : TOPHAT SOBRE OBJETOS OBSCUROS                    */
/*       im_Nwtophat_(ime,ims,size)                          */
/* TRANSF : TOPHAT SOBRE OBJETOS CLAROS                      */
/*       im_Nbtophat_(ime,ims,size)                          */
/*       ime : INPUT IMAGE (Binary Image)                    */
/*       ims : OUTPUT IMAGE (Binary Image)                   */
/*       size : SIZE OF THE STRUCTURE                        */
/*************************************************************/


void im_Nwtophat_(IMAGEN *ime, IMAGEN *ims, int size)
{

        im_Ngreyopen_(ime,ims,size)                ;
        im_greysub_(ime,ims,ims)                   ;

}
```

```
void im_Nbtophat_(IMAGEN *ime, IMAGEN *ims, int size)
{

        im_Ngreyclose_(ime,ims,size)                    ;
        im_greysub_(ims,ime,ims)                        ;

}


/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: FILTROS ALTERNADOS SECUENCIALES                                 */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA                            */
/*                   size   CONSTANTE                                           */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                             */
/*-------------------------------------------------------------------------------------------------------------*/

void im_fasw_(IMAGEN *ime,IMAGEN *ims, int size)
{
        int i           ;

        if(ime! =ims)
                im_copy_(ime,ims)       ;
        for(i=1 ; i<=size ; i++){
                im_Ngreyclose_(ims,ims,i)       ;
                im_Ngreyopen_(ims,ims,i)        ;
                }
}
void im_fasb_(IMAGEN *ime,IMAGEN *ims,int size)
{
        int i           ;

        if(ime! =ims)
                im_copy_(ime,ims)       ;
        for(i=1 ; i<=size ; i++){
                im_Ngreyopen_(ims,ims,i)        ;
                im_Ngreyclose_(ims,ims,i)       ;
                }
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: FILTROS  AUTOMEDIANOS, INFCENTRO, SUPCENTRO     */
/*       ENTRADAS:   ime  ESTRUCTURA IMAGEN NUMERICA                            */
/*                   size   CONSTANTE                                           */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                             */
/*-------------------------------------------------------------------------------------------------------------*/

void im_automed_(IMAGEN *ime,IMAGEN *ims,int size)
{

        IMAGEN *nw  ;

        nw  =  im_alloc_grey_(ime->sx,ime->sy,ime->lg)      ;

        im_Ngreyopen_(ime,ims,size)                     ;
```

```
        im_Ngreyclose_(ims,ims,size)                    ;
        im_Ngreyopen_(ims,ims,size)                     ;

        im_Ngreyclose_(ime,nw,size)                     ;
        im_Ngreyopen_(nw,nw,size)                       ;
        im_Ngreyclose_(nw,nw,size)                      ;

        im_greyinf_(ime,nw,nw)                                  ;
        im_greysup_(nw,ims,ims)                                 ;

        im_free_grey_(nw)                               ;

}
void im_centre_inf_(IMAGEN *ime,IMAGEN *ims,int size)
{

        im_Ngreyclose_(ime,ims,size)                    ;
        im_Ngreyopen_(ims,ims,size)                     ;
        im_Ngreyclose_(ims,ims,size)                    ;

        im_greyinf_(ime,ims,ims)                                ;

}

void im_centre_sup_(IMAGEN *ime,IMAGEN *ims,int size)
{


        im_Ngreyopen_(ime,ims,size)                     ;
        im_Ngreyclose_(ims,ims,size)                    ;
        im_Ngreyopen_(ims,ims,size)                     ;

        im_greysup_(ime,ims,ims)                                ;

}

BIMAGEN *im_MDFL_0_(int ires)
{

        BIMAGEN *bimage0,*bimage1       ;
        long indi, indi1        ;
        int i,j                         ;

        if(ires>7){
                printf("\n\a\a ERROR im_MDFL MAXIMA ITERACION 7 ");
                exit(0) ;
        }

        bimage0 = im_alloc_bin_(SIZEIMX,SIZEIMY,LEVELB)         ;
        bimage1 = im_alloc_bin_(SIZEIMX,SIZEIMY,LEVELB)         ;

        for(i=0 ; i<bimage0->sy ; i++)
                for(j=0 ; j<=i ; j++)
                        im_bwritepix_(bimage0,j,i,1)    ;
```

```
        im_bindisplay_(bimage0,1,0,0,0)         ;
        indi = 0        ;
        for(j=0 ; j<ires ; j++){
                im_bcopy_(bimage0,bimage1) ;
                i=0     ;
                while(im_area_(bimage1)!=0){
                        i++ ;
                        printf("            ") ; printf("\r %d ",i) ;
                        im_binerotri_(bimage1,2)        ;
                        }
                im_Nbinopentri_(bimage0,bimage1,i-1,2)          ;
                im_bininv_(bimage1,bimage1)             ;
                im_binand_(bimage1,bimage0)             ;
                im_bindisplay_(bimage0,1,0,0,1)                 ;
        }


        im_free_bin_(bimage1) ;

        return(bimage0) ;

}
```

LIBRERIA : PRIMIT3.C

CONTENIDO:

1) TRANSFORMACIONES USANDO ESTRUCTURAS DE DATOS MAS ELALORADAS.

a) ETIQUETADO RAPIDO
b) EXTRACCION DE CONTORNOS.
c) CALCULO DE VERTIENTES 8-VECINOS
d)CALUCLO DE VERTIENTES 4-VECINOS
e) CALCULO DEL WATERSHED 8-VECINOS
f) CALCULO DEL WATERSHED 4-VECINOS.

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include "c:\users\ivan\logiciel\strim.h"

extern void im_greydisplay_B_(IMAGEN *,int, int) ;

extern int gdsmode     ;
extern int im_flag     ;
int ifrx[8] = {0,-1,-1,-1,0,1,1,1}      ;
int ifry[8] = {1,1,0,-1,-1,-1,0,1}      ;

float cx[8] = {1.0,1.5,1.0,1.5,1.0,1.5,1.0,1.5}              ;
float cy[8] = {1.0,0.5,0.0,-0.5,-1.0,-0.5,0.0,0.5}     ;


int mrec[8][8] ={5,6,7,0,1,2,3,4,
                 6,7,0,1,2,3,4,5,
                 7,0,1,2,3,4,5,6,
                 0,1,2,3,4,5,6,7,
                 1,2,3,4,5,6,7,0,
                 2,3,4,5,6,7,0,1,
                 3,4,5,6,7,0,1,2,
                 4,5,6,7,0,1,2,3} ;
```

/*----------*/

```
/*      TRANSF: ETIQUETADO RAPIDO DE OBJETOS                                    */
/*       ENTRADAS:   ima  ESTRUCTURA IMAGEN BINARIA                             */
/*                   ime ESTRUCTURA IMAGEN NUMERICA                             */
/*      SALIDAS:     ims ESTRUCTURA IMAGEN NUMERICA                             */
/*-------------------------------------------------------------------------------*/

void im_labeling_(BIMAGEN *bime, IMAGEN *ime, IMAGEN *ims)
{
EIMAGEN *ftpr                ;
unsigned int far *pt,vmask   ;
unsigned int i,j,x,y,k       ;
long drap                    ;
unsigned char val            ;


ftpr = (EIMAGEN far *)farmalloc(1*sizeof(EIMAGEN));
ftpr->sx=bime->sx   ;       ftpr->sy=bime->sy   ;
ftpr->lg=bime->lg   ;
for(x=0 ; x<(ime->sy+2) ; x++){
        pt = (unsigned int far *)farmalloc((ime->sx+2)*sizeof(int))   ;
if(pt==0){
        printf("\n\a\a Error Allocacion im_alloc_grey");
        exit(1) ;
}
ftpr->p[x]=pt ;
}
        for(y=0 ; y<(ime->sy+2) ; y++)ftpr->p[y][0]=0 ;
        for(y=0 ; y<(ime->sy+2) ; y++)ftpr->p[y][ime->sx+1]=0 ;
        for(x=0 ; x<(ime->sx+2) ; x++)ftpr->p[0][x]=0 ;
        for(x=0 ; x<(ime->sx+2) ; x++)ftpr->p[ime->sy+1][x]=0 ;
        im_clear_(ime) ;


        for(y=1; y<=ime->sy ;y++)
                for(x=0 ; x<(ime->sx/16) ; x++){
                        vmask = 0x8000                  ;
                        for(k=1 ; k<=16 ; k++){
                        if((bime->p[y-1][x] & vmask)==0){
                                ftpr->p[y][x*16+k]= 0           ;
                        }else{
                                ftpr->p[y][x*16+k] =257*y+(x*16+k)      ;
                                }
                        vmask = vmask >> 1      ;
                        }
                }

        im_greysetmask0_(ime,bime,ime,255)              ;
/*****************************************************************************/

drap=1 ;
while(drap!=0){
drap = 0        ;

for(i=1; i<=ime->sy ;i++){
```

```
        for(j=1 ; j<=ime->sx ; j++){
                vmask = ftpr->p[i][j]           ;
                ftpr->p[i][j]=val_mask_max_(ftpr->p[i-1][j-1],ftpr->p[i-1][j],ftpr->p[i-
1][j+1],
                        ftpr->p[i][j-1],ftpr->p[i][j],ime->p[i][j]) ;
                drap += (vmask ^ ftpr->p[i][j])         ;
                }
}


for(i=(ime->sy); i>=1 ;i--){
        for(j=(ime->sx) ; j>=1 ; j--){
                vmask = ftpr->p[i][j]           ;
                ftpr->p[i][j] = val_mask_max_(ftpr->p[i+1][j+1],ftpr->p[i+1][j],ftpr-
>p[i+1][j-1],
                        ftpr->p[i][j+1],ftpr->p[i][j],ime->p[i][j])         ;
                drap += (vmask ^ ftpr->p[i][j])         ;
        }

}
}
/*********************************************************************/
        im_clear_(ims)                                          ;

val = 0 ;
for(y=1 ; y<=ims->sy ; y++)
for(x=1 ; x<=ims->sx ; x++){
        if((usi)ftpr->p[y][x]!=0)
        if(ims->p[(usi)(ftpr->p[y][x])/257][(usi)(ftpr->p[y][x]) % 257]==0){
        val = val +1   ;
        ims->p[(usi)(ftpr->p[y][x])/257][(usi)(ftpr->p[y][x]) % 257]=val ;

        }
}

/*********************************************************************/


/*********************************************************************/

        im_greybuild_(ime,ims)                                  ;
        im_greydisplay_(ims,0,0,1)                              ;
        getch() ;


for(x=0 ; x<(ime->sy+2) ; x++){
        pt=ftpr->p[x] ;
        farfree((unsigned int far *)pt)   ;
        }
farfree((EIMAGEN far *)ftpr)    ;

}

unsigned int val_mask_max_(usi am,usi bm,usi cm,usi dm,usi em, usc e)
```

```
{

        unsigned int minv       ;
        minv = am                       ;

        if((usi)minv < (usi)bm) minv = bm     ;
        if((usi)minv < (usi)cm) minv = cm     ;
        if((usi)minv < (usi)dm) minv = dm     ;
        if((usi)minv < (usi)em) minv = em     ;
        if((usc)e==0)minv = 0                           ;
        return(minv)    ;
}

/* CREA UNA ESTRUCTURA DE DATOS LACET */

lacet *cree_lacet_()
{

        lacet *nouv     ;


        nouv = (lacet *)malloc(1 * sizeof(lacet))       ;
        if(nouv){
                nouv->longueur = 0                      ;
                nouv->pre = nouv->der = 0               ;
        }else{
                printf("\n Error Alocacion Dinamica en cree_lacet") ;
                exit(0)                 ;
        }

        return(nouv)    ;

}

/*****************************************************************/
/*      TRANSF: SEGUIMIENTO DE CONTORNOS                         */
/*      kop     : Trama                                          */
/*      ngris   : Grey-Level Edge                                */
/*      image   : Input Image                                    */
/*****************************************************************/



lacet *im_lacet_(IMAGEN *image,usc ngris,usc kop,usc objhoy)
{


        int i,j,k,jtest,ii,z                    ;
        lacet *nouvl,*cree_lacet_()             ;
        noeud_t *cree_noeud_(),*nouvn,*p        ;
        int dirf,k2,if5,k1,k7,dir,ipoint,ci,cj      ;
```

```
        for(j=0 ; j<=(image->sy-1) ; j++)
                for(i=0 ; i<=(image->sx-1) ; i++)
                        if((usc)image->p[j][i]==(usc)ngris) goto salida ;
salida :
        printf("\n &&&& %d %d %u ",j,i,image->p[j][i]) ; if(getch()=='o')exit(0);

        image->p[j][i] = 255            ;

        ci = i ; cj = j ;
        if(j>=image->sy){
                printf("\a\a\n *** There are not objets *** ");
                exit(0)         ;
        }

        nouvl = cree_lacet_()           ;

        ajoute_(i,j,nouvl)              ;

        if(kop != 2)
                jtest = 7               ;
        else
                jtest = 6               ;
        p = nouvl->pre                  ;

        ii = 2                          ;


        do{
                i = p->x+ifrx[ii]               ;
                j = p->y+ifry[ii]               ;
                if((usc)image->p[j][i]<(usc)ngris)
                        ii += kop               ;

                if(ii>jtest)ii=0 ;
        printf("\n +++ %d %d %u ",j,i,image->p[j][i]) ; if(getch()=='o')exit(0);

        }while((usc)image->p[j][i]!=(usc)ngris)         ;
        image->p[j][i] = 255 ;
        im_greydisplay_(image,0,0,1)            ;

        printf("\n - %d %d %d %u ",j,i,ii,image->p[j][i]) ;
        if(getch()=='o')exit(0) ;



        ajoute_(i,j,nouvl)              ;

        dirf = ii ; k2 = 2 ; if5 = 0 ; k7 = 4 ; k1 = 1 ;

        p = (nouvl->pre)->suivant       ;


        do{
                do{
```

```
                    i = p->x + ifrx[mrec[k1][ii]]  ;
                    j = p->y + ifry[mrec[k1][ii]]  ;
        printf("\n -  %d %d %d %d %u ",j,i,ii,mrec[k1][ii],image->p[j][i]) ;
                    k1 + = kop                    ;
                    if(k1>7)k1=0          ;
                }while((usc)image->p[j][i]!=(usc)ngris)              ;
                image->p[j][i] = 255 ;
                ajoute_(i,j,nouvl)                ;
        if(getch() == 's')return(nouvl) ;
                p = p->suivant        ; k2++ ; dir=ii        ;
                ipoint = mrec[k1][ii]   ; k7++        ; ii=ipoint      ;
                k1 =1 ;
        }while(((usc)p->x != (usc)cj) || ((usc)p->y != (usc)ci))      ;

                im_greydisplay_(image,0,0,1) ;


        return(nouvl)           ;

}

/* CREA UNA ESTRUCTURA DE DATOS NODO       */
noeud_t *inst_noeud_(int valx,int valy,noeud_t *ptr)
{
                noeud_t *nouv          ;

                nouv = (noeud_t *)malloc(1 * sizeof(noeud_t))       ;

                if(nouv){
                        nouv->x = valx                ;
                        nouv->y = valy                ;
                        nouv->suivant = ptr ;
                }else{
                        printf("\a\n ERROR ALLOCACION DINAMICA EN inst_noeud()")
;
                        exit(0) ;
                }
return(nouv)    ;
}

/* CREA UNA ESTRUCTURA DE DATOS FILA */

fifo_t *cree_file_()
{
                fifo_t *nouv                                    ;


                nouv = (fifo_t *)malloc(1 * sizeof(fifo_t))     ;

                if(nouv){
                        nouv->longueur = 0                      ;
                        nouv->avant=nouv->arriere=0             ;
                }else{
                        printf("\a\n ERROR ALLOCACION DINAMICA EN cree_file_()");
```

```
                exit(0) ;
        }
return(nouv)    ;
}

/* METE UN VALOR EN UNA FILA */

BOOLEEN enfile_(int valx,int valy,fifo_t *file)
{
        noeud_t *nouv                   ;

        nouv = inst_noeud_(valx,valy,0)         ;

        if(nouv){
                if(file->longueur)
                        file->arriere->suivant = nouv;
                else
                        file->avant = nouv              ;
                file->arriere = nouv                    ;
                file->longueur++        ;
                return(VRAI)            ;
        }else
                return(FAUX)            ;

}

/* SACA UN VALOR DE UNA FILA */

int defiler_(int *valx,int *valy,fifo_t *file)
{

        noeud_t *temp       ;

        temp = file->avant      ;
        *valx = temp->x         ;
        *valy = temp->y         ;

        file->avant = temp->suivant ;

        file->longueur--        ;
        if(file->longueur==0)
                file->arriere = 0       ;
        free((noeud_t *)temp)   ;
return(0)       ;
}


/* CREA UNA FILA DE ESPERA JERARQUICA DE TAMAÑO log_h */

fifo_h_t *cree_fileh_(int long_h)
{

        fifo_h_t *nouv      ;
        fifo_t *p               ;
```

```
        int i                   ;


        nouv = (fifo_h_t *)farmalloc(1 * sizeof(fifo_h_t))      ;

        if(nouv==0){
                printf("\a\n ERROR ALLOCACION DINAMICA EN cree_fileh_()")
;
                exit(0) ;
        }

        p = (fifo_t *)farmalloc(long_h * sizeof(fifo_t)) ;

        if(p){
                nouv->long_h = long_h                   ;
                nouv->file_suivant = p                  ;

                for(i=0 ; i<nouv->long_h ; i++){
                        (p+i)->longueur = 0             ;
                        (p+i)->avant = (p+i)->arriere = 0   ;
                }

        }else{
                printf("\a\n ERROR ALLOCACION DINAMICA EN cree_fileh_()")
;
                exit(0) ;
        }

        return(nouv)            ;
}

/* ADICIONA UNA LISTA DE ESPERA A UNA FILA ESPERA JERARQUICA */

BOOLEEN free_fileh_(fifo_h_t *file_h)
{

        int i           ;

        farfree((fifo_t *)file_h->file_suivant)    ;

        farfree((fifo_h_t *)file_h)        ;

return(VRAI)    ;
}


/* INSERTA UN VLOR EN UNA LISTA */

BOOLEEN insere_(int valx,int valy,tete_t *liste)
{
```

```
        noeud_t *nouv, inst_noeud()   ;

        nouv=inst_noeud_(valx,valy,liste->pre) ;
        if(nouv){
                liste->pre = nouv      ;
                if(liste->longueur==0)
                        liste->der = nouv      ;
                liste->longueur++              ;
                return(VRAI)    ;
        }else
                return(FAUX)    ;
}


int ajoute_(int valx,int valy,tete_t *liste)
{
        noeud_t *nouv                  ;


        nouv=inst_noeud_(valx,valy,0)          ;
        if(nouv){
                if(liste->longueur)
                        liste->der->suivant=nouv      ;
                else
                        liste->pre = nouv             ;
        liste->der = nouv                             ;
        liste->longueur++              ;
        return(VRAI)                   ;
        }else
        return(FAUX)                   ;

}


/* SUPRIME UN VALOR DE UNA LISTA */

int supprime_(tete_t *liste)
{
        noeud_t *temp          ;

        temp = liste->pre      ;

        liste->pre = temp->suivant    ;

        liste->longueur--              ;

        if(liste->longueur ==0)
                liste->der = 0 ;

        free((noeud_t *)temp)  ;
        return(0)              ;
```

```
}

/* CREA UNA LISTA   */

tete_t *cree_liste_()
{
        tete_t *nouv    ;


        nouv = (tete_t *)malloc(1 * sizeof(tete_t))     ;
        if(nouv!=0){
                nouv->longueur = 0  ;
                nouv->pre = nouv->der = 0;
        }else{
                printf("\a\n ERROR ALOCACION DINAMICA en cree_liste")    ;
                exit(0) ;
        }


        return(nouv)    ;
}



/*-------------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CALCULO DE VERTIENTES EN UNA IMAGEN NUMERICA
                    A PARTIR DE UN MARCADOR (IMAGEN ETIQUETADA)
                     USANDO 8-VECINOS                                   */
/*       ENTRADAS:    image  ESTRUCTURA IMAGEN NUMERICA                 */
/*      SALIDAS:      marqueur ESTRUCTURA IMAGEN NUMERICA               */
/*-------------------------------------------------------------------------------------------------------------------*/

void im_vertientes_(IMAGEN *image,IMAGEN *marqueur)
{
        int i,j,k,long_h  ;
        usc valx, valy,val      ;
        int valxx,valyy         ;
        fifo_h_t  *fich_h       ;
        char b  ;

        valx=(usc)image->p[0][0]      ;
        long_h=(int)valx ;
        for(i=0 ; i<=image->sy ; i++)
                for(j=0 ; j<=image->sx ; j++)
                        {
                        valx=(usc)image->p[i][j] ;
                        if (long_h<(int)valx)
                        long_h=(int)valx        ;
                        }
        fich_h=cree_fileh_(long_h)      ;
        gdsmode = 0 ;
```

```
        for(i=0 ; i<marqueur->sy ; i++)
                for(j=0 ; j<marqueur->sx ; j++)
                {
                        b = (marqueur->p[i-1][j-1]&marqueur->p[i-1][j]&marqueur->p[i-
1][j+1]&
                           marqueur->p[i][j-1]&marqueur->p[i][j]&marqueur->p[i][j+1]&
                           marqueur->p[i+1][j-1]&marqueur->p[i+1][j]&marqueur-
>p[i+1][j+1])^marqueur->p[i][j];
                        if((usc)b>0)
                        {
                        valx =(usc)image->p[i][j] ; valx =valx-1;
                        enfile_((usc)j,(usc)i,&fich_h->file_suivant[(int)valx])    ;
                        };
                };
        for(i=0; i<fich_h->long_h ; i++)
        {
                while(fich_h->file_suivant[i].longueur)
                {
                defiler_(&valxx,&valyy,&fich_h->file_suivant[i]);
                if ((1<valxx)&&(valxx<image->sx)&&(1<valyy)&&(valyy<image->sy))
                {
                for(j=(valxx-1) ; j<=(valxx+1) ; j++)
                        for(k=(valyy-1) ; k<=(valyy+1) ; k++)
                                if (marqueur->p[k][j]==0)
                                {
                                        marqueur->p[k][j]=marqueur-
>p[valyy][valxx];
                                        valx =(usc)image->p[k][j] ; valx =valx-1;
                                        enfile_((usc)j,(usc)k,&fich_h-
>file_suivant[(int)valx])        ;
                                };
                };
                };
        };
}
/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CALCULO DE VERTIENTES EN UNA IMAGEN NUMERICA
                  A PARTIR DE UN MARCADOR (IMAGEN ETIQUETADA)
                   USANDO 4-VECINOS                                          */
/*       ENTRADAS:   image  ESTRUCTURA IMAGEN NUMERICA                      */
/*      SALIDAS:     marqueur ESTRUCTURA IMAGEN NUMERICA                    */
/*-------------------------------------------------------------------------------------------------------------*/


void im_vertientes_4_(IMAGEN *image,IMAGEN *marqueur)
{
        int i,j,k,long_h  ;
        usc valx, valy,val       ;
        int valxx,valyy          ;
        fifo_h_t  *fich_h        ;
        char b  ;

        valx=(usc)image->p[1][1]      ;
        long_h=(int)valx ;
```

```
        for(i=1 ; i<=image->sy ; i++)
                for(j=1 ; j<=image->sx ; j++)
                        {
                        valx=(usc)image->p[i][j] ;
                        if (long_h<(int)valx)
                        long_h=(int)valx        ;
                        }
        fich_h=cree_fileh_(long_h)      ;
        gdsmode = 0 ;
        for(i=1 ; i<=marqueur->sy ; i++)
                for(j=1 ; j<=marqueur->sx ; j++)
                {
                        b = (marqueur->p[i-1][j-1]&marqueur->p[i-1][j]&marqueur->p[i-
1][j+1]&
                           marqueur->p[i][j-1]&marqueur->p[i][j]&marqueur->p[i][j+1]&
                           marqueur->p[i+1][j-1]&marqueur->p[i+1][j]&marqueur-
>p[i+1][j+1])^marqueur->p[i][j];
                        if((usc)b>0)
                        {
                        valx =(usc)image->p[i][j] ; valx =valx;
                        enfile_((usc)j,(usc)i,&fich_h->file_suivant[(int)valx])    ;
                        };
                };
        for(i=0; i<=fich_h->long_h ; i++)
        {
                while(fich_h->file_suivant[i].longueur)
                {
                defiler_(&valxx,&valyy,&fich_h->file_suivant[i]);
                if ((1<valxx)&&(valxx<image->sx)&&(1<valyy)&&(valyy<image->sy))
                {
                for(j=(valxx-1) ; j<=(valxx+1) ; j++)
                        if (marqueur->p[valyy][j]==0){
                        marqueur->p[valyy][j]=marqueur->p[valyy][valxx];
                        valx =(usc)image->p[valyy][j] ; valx =valx;
                        enfile_((usc)j,(usc)valyy,&fich_h->file_suivant[(int)valx])        ;
                        }       ;
                for(k=(valyy-1) ; k<=(valyy+1) ; k++)
                        if (k!=valyy && marqueur->p[k][valxx]==0)
                                {
                                marqueur->p[k][valxx]=marqueur->p[valyy][valxx];
                                valx =(usc)image->p[k][valxx] ; valx =valx-1;
                                enfile_((usc)valxx,(usc)k,&fich_h->file_suivant[(int)valx])
;
                                if((int)valx<i)i=(int)valx;
                                };
                };
                };
        };
}

/*-------------------------------------------------------------------------------------------------*/
/*      TRANSF: CALCULO DEL WATERSHED EN UNA IMAGEN NUMERICA
                A PARTIR DE UN MARCADOR (IMAGEN ETIQUETADA)
                 USANDO 8-VECINOS                                   */
```

```
/*       ENTRADAS:    image  ESTRUCTURA IMAGEN NUMERICA            */
/*       SALIDAS:     marqueur ESTRUCTURA IMAGEN NUMERICA          */
/*-------------------------------------------------------------------------------------------------------*/


void im_watershed_(IMAGEN *image,IMAGEN *marqueur)
{
        int i,j,k,long_h  ;
        usc valx, valy,val      ;
        int valxx,valyy         ;
        fifo_h_t  *fich_h       ;
        char b  ;
        int iban        ;

        valx=(usc)image->p[1][1]     ;
        long_h=(int)valx ;
        for(i=1 ; i<=image->sy ; i++)
                for(j=1 ; j<=image->sx ; j++)
                        {
                        valx=(usc)image->p[i][j] ;
                        if (long_h<(int)valx)
                        long_h=(int)valx        ;
                        }
        fich_h=cree_fileh_(long_h+1) ;
        gdsmode = 0 ;
        for(i=1 ; i<=marqueur->sy ; i++)
                for(j=1 ; j<=marqueur->sx ; j++)
                {
                        b = (marqueur->p[i-1][j-1]&marqueur->p[i-1][j]&marqueur->p[i-
1][j+1]&
                           marqueur->p[i][j-1]&marqueur->p[i][j]&marqueur->p[i][j+1]&
                           marqueur->p[i+1][j-1]&marqueur->p[i+1][j]&marqueur-
>p[i+1][j+1])^marqueur->p[i][j];
                        if((usc)b>0)
                        {
                        valx =(usc)image->p[i][j] ; valx =valx;
                        enfile_((usc)j,(usc)i,&fich_h->file_suivant[(int)valx])    ;
                        };
                };
        for(i=0; i<=fich_h->long_h ; i++)
        {
                iban=fich_h->file_suivant[i].longueur;
                while(fich_h->file_suivant[i].longueur)
                {
                defiler_(&valxx,&valyy,&fich_h->file_suivant[i]);
                if ((1<=valxx)&&(valxx<=image->sx)&&(1<=valyy)&&(valyy<=image-
>sy))
                {
                val=marqueur->p[valyy][valxx] ;
                for(j=(valxx-1) ; j<=(valxx+1) ; j++)
                        for(k=(valyy-1) ; k<=(valyy+1) ; k++){
                                if(marqueur->p[k][j]!=val && marqueur->p[k][j]!=0 &&
(usc)marqueur->p[k][j]!=255)
                                {
```

```
                                        marqueur->p[valyy][valxx]=255;
                                }
                        }
/*              if((usc)marqueur->p[valyy][valxx]!=255) */
                for(j=(valxx-1) ; j<=(valxx+1) ; j++)
                        for(k=(valyy-1) ; k<=(valyy+1) ; k++){
                                if (marqueur->p[k][j]==0)
                                {
                                        marqueur->p[k][j]=val;
                                        valx =(usc)image->p[k][j] ; valx =valx;
                                        enfile_((usc)j,(usc)k,&fich_h-
>file_suivant[(int)valx])       ;
                                };
                        }

                };
                };
                if(iban!=0)im_greydisplay_(marqueur,0,0,1) ;
        };
}

/*-------------------------------------------------------------------------------------------------------------*/
/*      TRANSF: CALCULO DEL WATERSHED EN UNA IMAGEN NUMERICA
                A PARTIR DE UN MARCADOR (IMAGEN ETIQUETADA)
                 USANDO 4-VECINOS                                       */
/*       ENTRADAS:   image  ESTRUCTURA IMAGEN NUMERICA                  */
/*      SALIDAS:      marqueur ESTRUCTURA IMAGEN NUMERICA               */
/*-------------------------------------------------------------------------------------------------------------*/


void im_watershed_4_(IMAGEN *image,IMAGEN *marqueur)
{
        int i,j,k,long_h  ;
        usc valx, valy,val      ;
        int valxx,valyy         ;
        fifo_h_t  *fich_h       ;
        char b  ;
        int iban        ;

        valx=(usc)image->p[1][1]     ;
        long_h=(int)valx ;
        for(i=1 ; i<=image->sy ; i++)
                for(j=1 ; j<=image->sx ; j++)
                        {
                        valx=(usc)image->p[i][j] ;
                        if (long_h<(int)valx)
                        long_h=(int)valx        ;
                        }
        fich_h=cree_fileh_(long_h+1) ;
        gdsmode = 0 ;
        for(i=1 ; i<=marqueur->sy ; i++)
                for(j=1 ; j<=marqueur->sx ; j++)
                {
```

```
                        b = (marqueur->p[i-1][j]&marqueur->p[i][j-1]&marqueur-
>p[i][j]&marqueur->p[i][j+1]&
                            marqueur->p[i+1][j])^marqueur->p[i][j];
                        if((usc)b>0)
                        {
                        valx =(usc)image->p[i][j] ; valx =valx;
                        enfile_((usc)j,(usc)i,&fich_h->file_suivant[(int)valx])    ;
                        };
                };
        for(i=0; i<=fich_h->long_h ; i++)
        {
                iban=fich_h->file_suivant[i].longueur;
                while(fich_h->file_suivant[i].longueur)
                {
                defiler_(&valxx,&valyy,&fich_h->file_suivant[i]);
                if ((1<=valxx)&&(valxx<=image->sx)&&(1<=valyy)&&(valyy<=image-
>sy))
                {
                val=marqueur->p[valyy][valxx] ;
                for(k=(valyy-1) ; k<=(valyy+1) ; k++)
                        for(j=(valxx-1) ; j<=(valxx+1) ; j++){
                        if((k==(valyy-1)&&j==valxx) || (k==valyy) || (k==(valyy+1)
&& j==valxx))
                                if(marqueur->p[k][j]!=val && marqueur->p[k][j]!=0 &&
(usc)marqueur->p[k][j]!=255)
                                {
                                        marqueur->p[valyy][valxx]=255;
                                }
                        }
/*              if((usc)marqueur->p[valyy][valxx]!=255) */
                for(j=(valxx-1) ; j<=(valxx+1) ; j++)
                        for(k=(valyy-1) ; k<=(valyy+1) ; k++){
                        if((k==(valyy-1)&&j==valxx) || (k==valyy) || (k==(valyy+1)
&& j==valxx))
                                if (marqueur->p[k][j]==0)
                                {
                                        marqueur->p[k][j]=val;
                                        valx =(usc)image->p[k][j] ; valx =valx;
                                        enfile_((usc)j,(usc)k,&fich_h-
>file_suivant[(int)valx])       ;
                                        };
                        }

                };
                };
                if(iban!=0)im_greydisplay_(marqueur,0,0,1) ;
        };
}
```