

```

    _setvideomode(_ERESNOCOLOR);
    break;
case _VGA:
case _OVGA:
case _MCGA:
    _setvideomode(_VRES2COLOR);
    break;
case _HGC:
    _setvideomode(_HERCMONO);
    break;
default:
    printf("Este programa no rueda con esta tarjeta de video");
    exit(0);
}
_getvideocfg(&mymode);
maxx = mymode.numxpixels - 1;
maxy = mymode.numypixels - 1;
}

void draw_lines(void)
{
/*divide pantalla en +-x en punto 10 y +-y en punto 500*/
_setvieworg(newx(40),newy(960));/*declara pantalla de -40, 960 en X y de -960 a 40 en Y*/
_moveto(newx(-40),0);/*mueve cursor de trazo a izq.*/
_lineto(newx(960),0);/*traza linea desde izq. hasta der.*/
_moveto(0,newy(-960));/*mueve cursor de trazo hacia arriba*/
_lineto(0,newy(40));/*traza linea desde arriba hasta abajo*/
}

int newx (float xcoord)
{
int nx;
float tempx;
tempx = ((float) maxx) / 1000.0;
tempx = ((float) xcoord) * tempx + 0.5;
return((int)tempx);
}

int newy (float ycoord)
{
int ny;
float tempy;
tempy = ((float) maxy) / 1000.0;
tempy = ((float)ycoord) * tempy + 0.5;
return((int)tempy);
}

void end_program(void)
{
getch();
_setvideomode(_DEFAULTMODE);
}

```

## A7.2 PROGRAMA GENERADOR DE NUMEROS ALEATORIOS.

```
#include<float.h>
#include<stdlib.h>
#include<time.h>
#define IM 714025L
#define IA 1366L
#define IC 150889L
#define TABLE_LENGTH 97

static long seed=797;
static long randout;
static long table[TABLE_LENGTH];
static int table_initialized=0;

void pf_srand(long iseed)
{
    seed=iseed;
    table_initialized=0;
}

long pf_rand()
{
    int i;
    if(!table_initialized)
    {
        table_initialized=1;
        for(i=0;i<TABLE_LENGTH;i++)
        {
            seed=(IA*seed+IC)%IM;
            table[i]=seed;
        }
        seed=(IA*seed+IC)%IM;
        randout=seed;
    }
    i=(int)((double)TABLE_LENGTH*(double)randout/(double)IM);
    randout=table[i];
    seed=(IA*seed+IC)%IM;
    table[i]=seed;
    return randout;
}

float alefrac()
{
    float fracclon;
    fracclon=(float)(pf_rand()/(float)(IM-1L));
    fracclon=fracclon*2;
    return(fracclon);
}
```

## A7.3 PROGRAMA QUE SIMULA UNA RED CUALQUIERA

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

```

#include<stdlib.h>
#include<iostream.h>
#include<string.h>
#include<float.h>
#include<ejesgraf.h>//Archivo particular.

char auxc;

class nodored
{
public:
    nodored(int numnodcapaanter, float *vec_tipoparampesos);
    nodored();
    float calc_sall(float *dato_entr);
    void cambiointer(float *cambioparam);
    void cambioexter(float *cambiopeso);
    float dapainter(){return(param[0]);};
    void operator= (const nodored &otro);
    ~nodored();
private:
    int i, tiponodo, dimvecentrada;
    float *peso, *param, sal_nodo;
};

nodored::nodored(int numnodcapaanter, float *vec_tipoparampesos)
{
    dimvecentrada=numnodcapaanter;
    tiponodo=(int)vec_tipoparampesos[0];
    param= new float[2];
    param[0]=vec_tipoparampesos[1];
    param[1]=vec_tipoparampesos[2];
    peso= new float[numnodcapaanter];
    if(peso)
    {
        for(i=0;i<dimvecentrada;i++)
        {
            peso[i]=vec_tipoparampesos[i+3];
        }
    }
    else
    {
        cout<<"Memoria insuficiente para crear vector de pesos";
    }
}

nodored::nodored()
{
    peso= new float[1];
    param= new float[2];
}

nodored::~~nodored()
{
    delete [] peso;
    delete [] param;
}

void nodored::operator= (const nodored &otro)
{

```

```

dimvecentrada=otro.dimvecentrada;
tiponodo=otro.tiponodo;
delete [] param;
param= new float[2];
param[0]=otro.param[0];
param[1]=otro.param[1];
delete [] peso;
peso=new float[dimvecentrada];
if(peso)
{
for(i=0;i<dimvecentrada;i++)
{
peso[i]=otro.peso[i];
}
}
else
{
cout<<"Memoria Insuficiente";
}
}

void nodored::camblointer(float *cambloparam)
{
for(i=0;i<2;i++)
{
param[i]=param[i]*cambloparam[i];
}
}

void nodored::cambioexter(float *camblopeso)
{
for(i=0;i<dimvecentrada;i++)
{
peso[i]=camblopeso[i];
}
}

float nodored::calc_sall(float *dato_entr)
{
int suichemultiplo=0;
float suma1=0, suma2=0, ancho, centro, escsalida, escentrada;
for(i=0;i<dimvecentrada;i++)
{
suma1+=peso[i]*dato_entr[i];
suma2+=dato_entr[i];
}
switch(tiponodo)
{
case 10: //Lineal con factor de escalado acotado a [-1,1].
escentrada=param[0];
if(suma1<0)
sal_nodo=__max(-1,escentrada*suma1);
if(suma1>=0)
sal_nodo=__min(1,escentrada*suma1);
break;
case 11://Lineal sin escalado ni acotado.
sal_nodo=suma1;
break;
}
}

```

```

case 20: //Conj. Pertenencia Fuzzy Campana Gaussiana.
ancho=20/(pow(param[0],2));
centro=param[1];
sal_nodo=exp(-ancho*pow((suma1-centro),2));
if(sal_nodo<1e-4)
    sal_nodo=0;
break;
case 21: //Conj. Pertenencia Fuzzy triangular isocoles.
ancho=param[0];
centro=param[1];
if(suma1<(centro-ancho/2)||suma1>(centro+ancho/2))
{
    sal_nodo=0;
}
else
{
    sal_nodo=1-(2*fabs(suma1-centro))/ancho;
}
break;
case 30: //Concrecion por metodo de las Alturas con escalado a la salida.
escsalida=param[0];
if(suma2==0)
    sal_nodo=0; //No se puede obtener una conclusion!
else
    sal_nodo=escsalida*(suma1/suma2);
break;
case 40: //T-Norma particular, por producto.
sal_nodo=1;
for(i=0;i<dimvecentrada;i++)
{
    suma1=peso[i]*dato_entr[i];
    if(peso[i]!=0)
    {
        suichemultiplo=1;
        sal_nodo=sal_nodo*suma1;
    }
}
if(sal_nodo<=1e-4)
    sal_nodo=0;
if(suichemultiplo==0)
    sal_nodo=0;
break;
case 41: //T-Norma por minimo.
sal_nodo=1;
for(i=0;i<dimvecentrada;i++)
{
    suma1=peso[i]*dato_entr[i];
    if(peso[i]!=0)
    {
        suichemultiplo=1;
        sal_nodo=__min(sal_nodo,suma1);
    }
}
if(sal_nodo<=1e-4)
    sal_nodo=0;
if(suichemultiplo==0)
    sal_nodo=0;
break;

```

```

    }
    return(sal_nodo);
}

class capared
{
public:
    capared(int numerocapa, FILE *red_config);
    capared();
    void capared::operator= (const capared &otro);
    float *vec_salida(float *vec_datos);
    void cambiointercap(int numidennodo, float *cambioparam);
    void cambioexterno(int numidennodo, float *cambio peso);
    float sacaparam(int numidennodo);
    ~capared();
private:
    int numcapasred, numcapactual, numnodoscapanter, numnodoscapactual;
    nodored *nodoscapa;
    float *salida;
};

capared::capared(int numerocapa, FILE *red_config)
{
    int i, j, aux, numsalto=0;
    float *vec_confignodo;
    char auxc;
    fscanf(red_config, "%d\n", &numcapasred); //lee numero de capas de la red.
    if(numcapasred < numerocapa)
    {
        cout << "Numero de capa no existente ";
        auxc=getch();
    }
    else
    {
        numcapactual=numerocapa;
        for(i=0; i<=numcapactual; i++)
        {
            fscanf(red_config, "%d ", &aux);
            numsalto+=aux;
            if(i==numcapactual)
            {
                numnodoscapactual=aux;
                if(numcapactual==0)
                {
                    numnodoscapanter=numnodoscapactual;
                }
                numsalto-=aux;
                if(numcapactual==(numcapasred-1))
                {
                    numsalto--;
                }
            }
            if(i==numcapactual-1)
            {
                numnodoscapanter=aux;
            }
        }
        movlinarch(numsalto+1, red_config);
    }
}

```

```

salida= new float[numnodoscapactual];
nodoscapa= new nodored[numnodoscapactual];
vec_conflgnodo= new float[numnodoscapanter+3];
for(j=0;j<numnodoscapactual;j++)
{
    for(i=0;i<(numnodoscapanter+3);i++)
    {
        fscanf(red_config,"%f",&vec_conflgnodo[i]);
    }
    nodored nodogenerico(numnodoscapanter, vec_conflgnodo);
    nodoscapa[j]= nodogenerico;
}
delete [] vec_conflgnodo;
}
}

```

```

capared::capared()

```

```

{
    nodoscapa= new nodored[1];
}

```

```

capared::~~capared()

```

```

{
    delete [] nodoscapa;
    delete [] salida;
}

```

```

void capared::operator= (const capared &otro)

```

```

{
    int i;
    numcapasred=otro.numcapasred;
    numcapactual=otro.numcapactual;
    numnodoscapanter=otro.numnodoscapanter;
    numnodoscapactual=otro.numnodoscapactual;
    delete [] nodoscapa;
    nodoscapa= new nodored[numnodoscapactual];
    if(nodoscapa)
    {
        for(i=0;i<numnodoscapactual;i++)
        {
            nodoscapa[i]=otro.nodoscapa[i];
        }
    }
    else
    {
        cout<<"Memoria insuficiente para crear nodos";
    }
}

```

```

void capared::cambiointercap(int numidennodo, float *cambioparam)

```

```

{
    int i;
    nodoscapa[numidennodo].cambiointer(cambioparam);
}

```

```

void capared::cambioexterno(int numidennodo, float *cambiospeso)

```

```

{
    int i;

```

```

    nodoscapa[numidennodo].cambioexter(cambiopeso);
}

float capared::sacaparam(int numidennodo)
{
    float painter;
    painter=nodoscapa[numidennodo].dapainter();
    return(painter);
}

float *capared::vec_salida(float *vec_dato)
{
    int i;
    delete [] salida;
    salida= new float[numnodoscapactual];
    if(salida)
    {
        for(i=0;i<numnodoscapactual;i++)
        {
            salida[i]=nodoscapa[i].calc_sall(vec_dato);
        }
    }
    else
    {
        cout<<"Mem.insuficiente Vector salida"<<numnodoscapactual;
    }
    return (salida);
}

class red
{
public:
    red(int numerodecapas, FILE *archivored);
    int dimsalida() {return(dimensionvecsal);};
    float *factpertenencia() {return(vecfacper);};
    float *propagadato(float *vectordato);
    void apreninternodo(int numidencapa, int numidennodo, float *cambioparam);
    float daparaminter(int numidencapa, int numidennodo);
    ~red();
private:
    int numcapasred, dimensionvecsal, dimensionvecinter;
    float *vecsalred, *vecfacper;
    capared *capasdered;
};

red::red(int numerodecapas, FILE *archivored)
{
    int i, aux;
    dimensionvecinter=1;
    fscanf(archivored, "%d\n", &numcapasred);
    if(numcapasred!=numerodecapas)
    {
        cout<<"Archivo de red no corresponde con red solicitada";
        cin>>aux;
    }
    else
    {
        capasdered = new capared[numerodecapas];
    }
}

```

```

rewind(archivored);
for(i=0;i<numerodecapas;i++)
{
    capared capagenerica(i,archivored);
    capasdered[i]= capagenerica;
    rewind(archivored);
}
mov/inarch(1, archivored);
for(i=0;i<numcapasred;i++)
{
    fscanf(archivored, "%d ", &aux);
    if(aux>dimensionvecinter)
    {
        dimensionvecinter=aux;
    }
    ddimensionvecsal=aux;
}
rewind(archivored);
}
vecsalred = new float[dimensionvecinter];
}

red::~~red()
{
    delete [] capasdered;
    delete [] vecsalred;
}

float *red::propagadato(float *vectordato)
{
    int i;
    float *datopropagado;
    datopropagado = new float[dimensionvecinter];
    datopropagado = vectordato;
    delete [] vecsalred;
    vecsalred = new float[dimensionvecinter];
    for(i=0;i<numcapasred;i++)
    {
        vecsalred = capasdered[i].vec_salida(datopropagado);
        datopropagado = vecsalred;
    }
    delete [] datopropagado;
    return(vecsalred);
}

void red::apreninternodo(int numidencapa, int numidennodo, float *cambioparam)
{
    int i;
    capasdered[numidencapa].cambiointercap(numidennodo, cambioparam);
}

float red::daparaminter(int numidencapa, int numidennodo)
{
    float paramdado;
    paramdado=capasdered[numidencapa].sacaparam(numidennodo);
    return(paramdado);
}

```