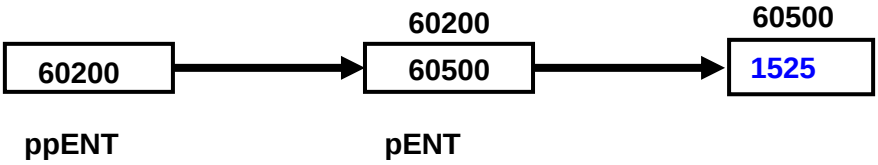
PUNTEROS DOBLES

Un puntero doble es una variable cuyo contenido es una dirección donde a su vez se halla almacenada otra dirección:



Analizar las siguientes notaciones:

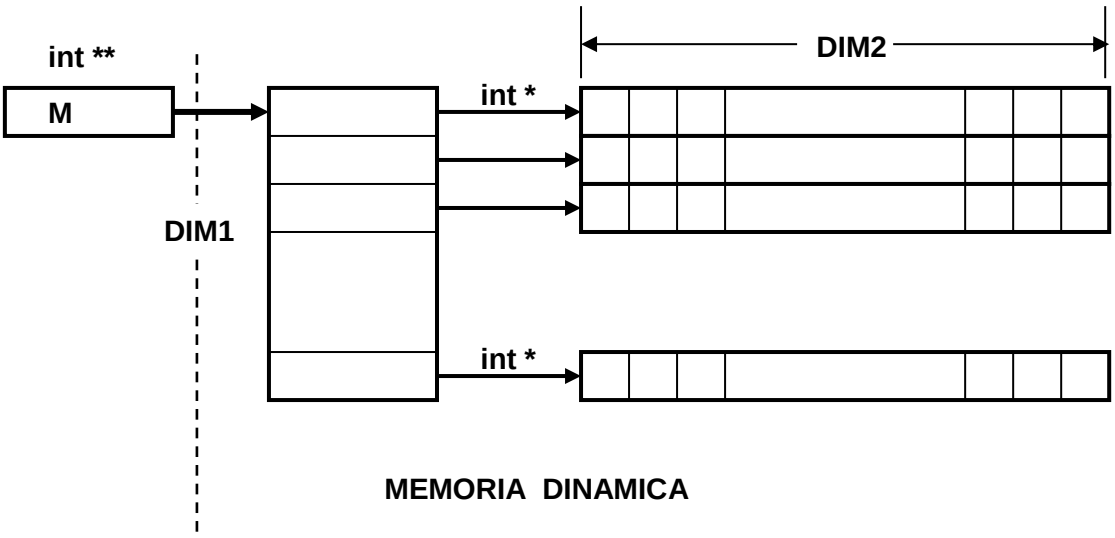
cprintf("%u",ppENT) --> muestra 60200
cprintf("%u",*ppENT) --> muestra 60500
cprintf("%d",ppENT) --> muestra 1525**

O sea que la primera indirección referencia la dirección donde se halla el dato, y la segunda el dato en sí: 1525.

```
//----- EjeClase4_1.cpp----
#include <conio.h>
#include <stdlib.h>

void main()
{
  int A;  //variable
  int *pA; //puntero
  int **ppA;//puntero doble

  A=5;
  cprintf("dato de A: %d\r\n",A);
  pA=&A;
  cprintf("direccion de pA: %d, dato de *pA: %d\r\n",pA,*pA);

  ppA=&pA;
  cprintf("direccion de ppA:%u, dato de *ppA:%u\r\n",ppA,*ppA);
  cprintf("lo que apunta el dato de **ppA: %d \r\n",**ppA);

  **ppA=25;

  cprintf("variable a: %d",A);
}
```

Arreglos bidimensionales (Matrices) en memoria dinámica

Si el contenido en sí de **ppENT** es la dirección donde se halla otro puntero, bien podría ser un arreglo de apuntadores en lugar de uno sólo:



Anteriormente habíamos creado una matriz semidinámica utilizando un arreglo estático de apuntadores, y recién a partir de allí entrábamos en la zona dinámica para hacer las reservas de DIM2 domicilios para cada elemento del arreglo de apuntadores, ahora toda la matriz estará en memoria dinámica.

Para crear una matriz dinámica tenemos que tener en cuenta lo siguiente:

- **Asignar memoria a un arreglo de punteros**, cuyos elementos referenciaran cada una de las filas de la matriz que deseamos crear.

- **Asignar memoria para cada una de las filas.** El número de elementos de cada fila puede ser variable. Con esto estaríamos reservando memoria para las “columnas de la matriz”

Una vista rápida seria:

```
const DIM1= 5;
const DIM2= 6;
...
int **matriz;
matriz = (int **) ReservarMemoria(DIM1*sizeof(int *));

for (i=0;i<FILAS;i++)
        matriz[i] = (int *) ReservarMemoria (DIM2*sizeof(int));
        for(j=0;j<DIM2;j++)
```

```
matriz[ i ][ j ] = 50 + random(50);
```

Según el código anterior podemos decir que cada fila puede tener una cantidad de columnas de diferentes tamaños. El uso del for, es para poder acceder a cada fila (matriz[i]) asignándole memoria.

Nótese que la reserva de memoria para las filas, lleva como argumento: **DIM1*sizeof(int *)** en tanto que la segunda: **DIM2*sizeof(int)**. La primera es una reserva de memoria para punteros a enteros, y la segunda una reserva sólo de enteros.

Podemos acceder a cualquier domicilio en la forma clásica subindexada: **M[i][j]** como en los arreglos estáticos.

```
//------------------------EjeClase4_2.cpp
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
#include <process.h>

const DIM1 = 7;
const DIM2 = 8;

void *ReservarMemoria(int TotBytes);
void main()
{
  int **M;
  int    i,j;

  clrscr( ); highvideo( ); randomize( );

  // --- Genera y asigna la matriz dinámica -------------------------

  M = (int **)ReservarMemoria(DIM1*sizeof(int *));
  for(i=0;i<DIM1;i++) {
            M[ i ] = (int *)ReservarMemoria(DIM2*sizeof(int));
            for(j=0;j<DIM2;j++)
                     M[ i ][ j ] = 50 + random(50);
  }

  // --- Muestra por pantalla los valores asignados ----------------

  for(i=0;i<DIM1;i++) {
            for(j=0;j<DIM2;j++)
                     cprintf("%4d",M[ i ][ j ]);
            cprintf("\r\n");
  }

  getch( );
}
```

```
void *ReservarMemoria(int TotBytes)
{
void *pAux;

  if((pAux=malloc(TotBytes))==NULL) {
    cprintf("No pudo reservar memoria dinámica");
    getch( ); exit(1);
  }
  return(pAux);
}
```

Punteros dobles como parámetros por referencia a funciones.

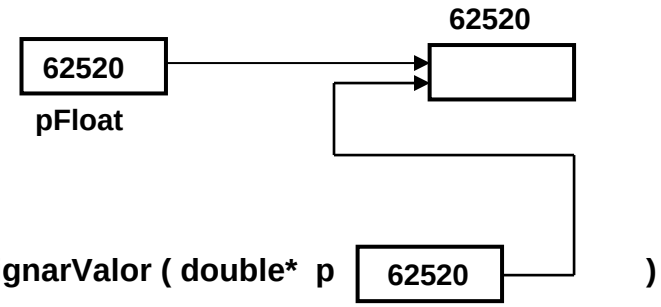
Existen dos situaciones en las cuales podemos pasarle un puntero a una función:

a) Para asignarle un contenido.
b) Para asignarle una dirección.

En el primer caso no es necesario un puntero doble, puesto que al pasarlo por valor estamos pasando una **copia** de dicho puntero **que ya tiene asignada una dirección**.

```
void AsignarValor(double*);

void main( )
{
   double*  pDouble;

   pDouble = (double*)ReservarMemoria(sizeof(double));
   AsignarValor(pDouble);
   cprintf("Valor asignado = %lf\r\n",*pDouble);
   getch( );
}

void AsignarValor(double* p)
{ *p = 9812 }
```

Esto podemos visualizarlo mucho mejor a través de un esquema:



O sea se hace una copia pFloat con el valor ya asignado por la función **AsignarMemoria( )**. Esto significa que cuando ejecute: ***p = 9812** en realidad se esté asignando en la dirección del dato original

.

NOTA

Si dentro de la función hiciéramos:

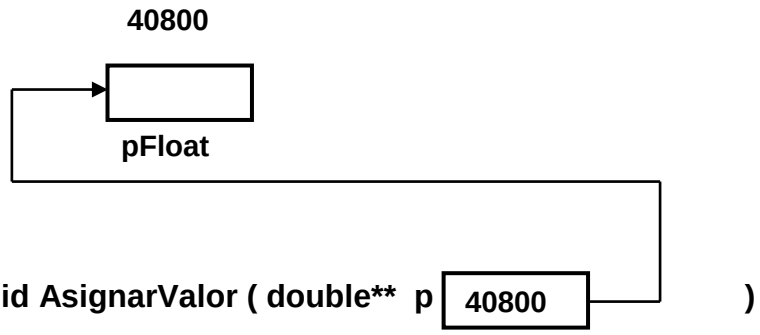
```
p=(double*)ReservarMemoria(sizeof(double));
*p = 9812;
```

Estaríamos cambiando la dirección original de **p** pero sin que ello repercuta en el main, y al retornar, el valor 9812 se habría perdido.

En cambio si tuviésemos que asignar una **dirección** a pFloat a través de dicha función, tendríamos que trabajar directamente sobre **p** y no sobre *p:

```
void AsignarValor(double**);

void main( )
{
  double*  pDouble;

  pDouble = (double*)ReservarMemoria(sizeof(double));
  AsignarValor(&pDouble);
  cprintf("Valor asignado = %lf\r\n",*pDouble);
  getch( );
}

void AsignarValor(double** p)
{
    *p = (double*)ReservarMemoria(sizeof(double));
   **p = 9812;//le asingamos el valor numerico
}
```

Nuevamente un esquema operativo nos permitirá visualizar detalladamente qué es lo que ha ocurrido con este cambio de sintaxis:



Ahora le estamos pasando a la función &pFloat, o sea la dirección del apuntador en sí (que aún no ha recibido ninguna dirección válida). De esta manera al hacer:

***p = (double *)ReservarMemoria( )**

estamos actuando directamente en el **contenido** de pFloat, la dirección válida del mismo. Y al ejecutar:

****p = 9812;**

Estamos actuando en la dirección apuntada por pFloat.

Los ejemplos completos son EjeClase4_3.cpp y EjeClase4_4.cpp

Listas enlazadas utilizando funciones

En la clase anterior cuando estudiamos las listas enlazadas, resolvimos la generación de un nodo nuevo dentro del propio main( ). Con los conocimientos actuales de punteros podemos mejorar esa rutina, asignando espacio a través de funciones.
Hay varias formas:

1. Usando una función que devuelve un puntero a una estructura

   ```
   TNodo *NodoNuevo()
   {
     TNodo *Direcc;
     Direcc=(TNodo *)ReservarMemoria(sizeof(TNodo));
     return(Direcc);
   }
   ```

   y al invocarla desde el main( ), le enviamos: **Aux->Next=NuevoNodo()**

2. Usando una función que recibe un puntero doble a una estructura

   ```
   void NuevoNodo(TNodo **p)
   {
           *p = (TNodo*)ReservarMemoria(sizeof(TNodo));
   }
   ```

y al invocarla desde el main( ), le enviamos: **NuevoNodo(&Aux->Next)**

3. Usando una función que recibe el inicio de la lista enlazada y se encarga de ir "enganchando" nodos nuevos:

```
TNodo *CrearNodo(TNodo **Start)
{
  TNodo *pAux;

  if(*Start==NULL) {
     *Start=NuevoNodo();
     (*Start)->next=NULL;
     return(*Start);
  }
  else {
    pAux=*Start;
   while(pAux->next){
    pAux=pAux->next; //este while es para llegar al ultimo
nodo
   }

   pAux->next=NuevoNodo();
   pAux=pAux->next;
   pAux->next=NULL;
   return(pAux);
  }
}
```

En la cual se detectan una serie de eventos: como si la lista debe o no ser inicializada dando un valor por primera vez a Start, o se trata de un nodo iésimo en cuyo caso debe recorrerse la lista ya elaborada hasta hallar el nodo final y recién allí agregar el próximo.
Esta función devuelve el ultimo nodo listo para ser usado

Los ejemplos completos son EjeClase4_5.cpp, EjeClase4_6.cpp y EjeClase4_7.cpp

Arreglo de cadenas de caracteres utilizando funciones

Este es un ejemplo donde se utiliza punteros dobles:

```
//----------------EjeClase4_9.cpp
#include<conio.h>

const DIM = 5;

void MostrarCadena ( char **);

// ---------------------------------------------------------

void main()
{
  char *MT[] = {"TUCUMAN"   ,
                "SALTA"     ,
                "JUJUY"     ,
                "CATAMARCA" ,
                "SANTIAGO"    };
  int    i;

  clrscr(); highvideo();

  for(i=0;i<DIM;i++) MostrarCadena(&MT[i]);

  getch();
}

// ---------------------------------------------------------

void MostrarCadena(char **Cad)
{ cprintf("%s\r\n",*Cad); }
```

La función de usuario **MostrarCadena()** recibe la dirección de un domicilio cuyo contenido es la dirección de una cadena de caracteres. De ahí que debe estar definida como:

void MostrarCadena(char **p);

Al invocarla le pasamos **&MT[i]**, o sea un apuntador doble.

Ejemplo de manejo de matrices dinámicas con funciones

El programa muestra una matriz identidad generada a partir de un valor dado por el usuario.

```
//----------------EjeClase4_9.cpp
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
```

```
#include <malloc.h>

void *ReservarMemoria(int TotBytes);
int ** asignar_matriz(int x); // asigna la memoria de la matriz
void diagonal_matriz(int **m,int x); // carga la diagonal principal con 1
void imprimir_matriz(int **m,int x); // imprime la matriz

void main(void){

    int n; // dimensión de la matriz
    int i,j; // contadores
    int **matriz=NULL; // puntero doble para la matriz dinámica
    clrscr( ); highvideo( );
    cprintf("Introduce el tamanio de la matriz:");
    scanf("%d",&n);

    matriz=asignar_matriz(n);
    diagonal_matriz(matriz,n); // paso de la matriz a funciones
    imprimir_matriz(matriz,n); // paso de la matriz a funciones

}

void *ReservarMemoria(int TotBytes)
{
void *pAux;

  if((pAux=malloc(TotBytes))==NULL) {
    cprintf("No pudo reservar memoria dinámica");
    getch( ); exit(1);
  }
  return(pAux);
}
// asigna la memoria de la matriz

int ** asignar_matriz(int x){

     int i;
     int **m;

     m=(int **)ReservarMemoria(x*sizeof(int *)); // se reserva memoria  para la matriz de
x filas
     //que contiene direcciones de memoria a las segundas dimensiones.

     for (i=0;i<x;i++){
        m[i]=(int  *)ReservarMemoria(x*sizeof(int));  //  se  reserva  memoria  para  las  x
columnas
     }

      // en memoria ya tenemos reservado espacio para una matriz de x por x --> m[x][x]
         return m; // retorno de un puntero doble
    }

// rellena de 1s la diagonal
```

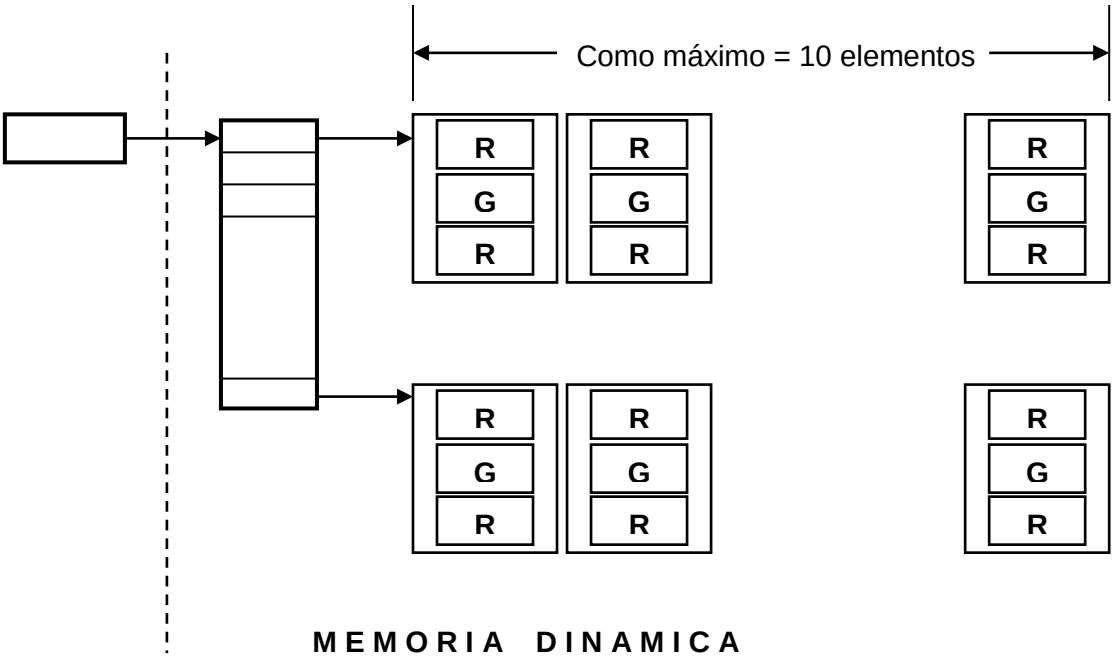
```
void diagonal_matriz(int **m,int x){

    int i,j;

    for (i=0;i<x;i++)
       for (j=0;j<x;j++)
           if (i==j)
              m[i][j]=1;
           else
              m[i][j]=0;

    }

// imprime la matriz

void imprimir_matriz(int **m,int x){

    int i,j;
    cprintf("\nIMPRIMIENDO MATRIZ\r\n");
    cprintf("\n==================\r\n");

    // manejo del puntero doble recibido como si fuese una matriz

    for (i=0;i<x;i++){
           for (j=0;j<x;j++){
                          if (i==j)
                                 textcolor(LIGHTRED);//para resaltar la diagonal principal
                          else
                                                textcolor(WHITE);
                cprintf("%d ",m[i][j]);
              }
           cprintf("\r\n");
   }

} // rellena de 1s la diagonal

void diagonal_matriz(int **m,int x){

    int i,j;

    for (i=0;i<x;i++)
       for (j=0;j<x;j++)
           if (i==j)
              m[i][j]=1;
           else
              m[i][j]=0;

    }

// imprime la matriz

void imprimir_matriz(int **m,int x){
```

```
    int i,j;
    cprintf("\nIMPRIMIENDO MATRIZ\r\n");
    cprintf("\n==================\r\n");

    // manejo del puntero doble recibido como si fuese una matriz

    for (i=0;i<x;i++){
          for (j=0;j<x;j++)
                cprintf("%d ",m[i][j]);
          cprintf("\r\n");
   }

}
```

Ejemplo de manejo de matrices dinámicas con estructuras

Una fotografía color puede considerarse como un arreglo bidimensional de **F**x**C** (filas, columnas) en el cual cada domicilio contiene 3 valores o códigos de color: **R**, **G**, **B**, correspondientes al **R**ed, **G**reen, **B**lue (Rojo, Verde y Azul). Esos códigos son valores enteros comprendidos entre 0 y 255, o sea que se requiere un byte como máximo para almacenar cada uno de ellos.

Genere la estructura necesaria para trabajar con una figura que, como máximo, tenga 5x10. Los valores de R, G, B serán generados aleatoriamente según los limites especificados.



Una vez asignados todos los puntos de la fotografía, tomara la dirección de cada fila y se la pasará a una función denominada:

MostrarTabla( )

que realizará la siguiente tarea:

```
   FILA 1
   145 238 .....................25
   0   100 .....................250  (deben aparecer 10 columnas)
   80  48  .....................125


   FILA 2
   idem
```

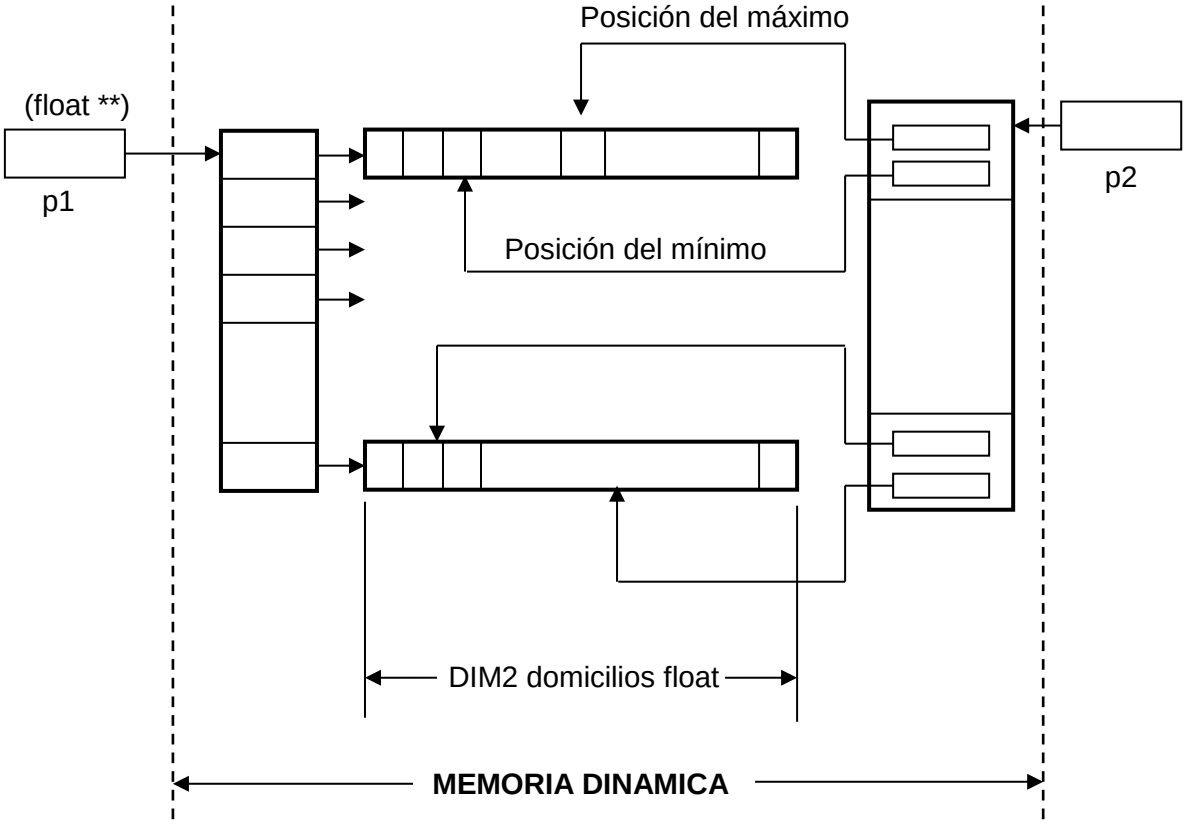
Para poder lograr este tipo de presentación deberá utilizar un par de variables denominadas "static" que poseen la particularidad de mantener su último valor al reingresar en el próximo llamado de la función.

//---------- EjeClase4_10.cpp ----------

```
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>

const ESC = 27;

typedef unsigned int BYTE;
typedef struct  { BYTE R; BYTE G; BYTE B; }TColor;

void *ResMem       ( int Total         );
void  MostrarTabla ( TColor **p, int C );

// ------------------------------------------------------
void main()
{
  TColor    **Foto;
  int         F     = 5; // dimension horizontal de la foto.
  int         C     = 10; // dimension vertical de la foto.
  int         i,j;

  clrscr(); highvideo(); randomize();

  Foto=(TColor **)ResMem(F*sizeof(TColor *));

  // --- asigna la tabla de colores -----------------

  for(i=0;i<F;i++){
      Foto[i]=(TColor *)ResMem(C*sizeof(TColor));
      for(j=0;j<C;j++){
        Foto[i][j].R=random(256);
        Foto[i][j].G=random(256);
        Foto[i][j].B=random(256);
      }
  }

  for(i=0;i<F;i++)
```

```
    MostrarTabla(&Foto[i],C);

  getch();

}

// ------------------------------------------------------------
void *ResMem(int Total)
{
  void *p;
  if((p = malloc(Total))==NULL) {
      cprintf("NO PUDO RESERVAR MEMORIA");
      getch(); exit(1);
  }
  return(p);
}

// ------------------------------------------------------------
void MostrarTabla(TColor **p, int C)
{
        int i;
        int Col  = 2;
 static int Fila = 1;
 static int fila_mostrar = 1;
 textcolor(LIGHTRED);
 gotoxy(2,Fila); cprintf("FILA %d - %d",fila_mostrar,Fila);
 textcolor(WHITE);
 for(i=0;i<C;i++,Col+=4){
    gotoxy(Col,Fila+1);
    cprintf("%d",(*p)[i].R);
    gotoxy(Col,Fila+2);
    cprintf("%d",(*p)[i].G);
    gotoxy(Col,Fila+3);
    cprintf("%d",(*p)[i].B);
 }
 Fila+=5;
 fila_mostrar++;
 if(getch()==ESC)
    exit(1);
}
// ------------------------------------------------------------
```

Otro ejemplo de matrices dinámicas con estructuras



Un puntero doble a float, llamado "p1", será el encargado de realizar una reserva dinámica matricial de 10 x 10 (DIM1=10 / DIM2=10), y un puntero simple p2 de TExtr (tipo extremos) señalará el comienzo de otra reserva dinámica de DIM1 domicilios. El tipo TExtr es el siguiente:

struct TExtr { float *Max; float *Min; };

Ambas reservas serán implementadas a través de la función de usuario:

void *ReservarMemoria(int Total)

Ahora desde el main() proceda a cargar aleatoriamente la reserva bidimensional con valores flotantes comprendidos entre 50.00 y 100.00

A través de una función de usuario denominada **ProcesarDatos( )** que tomará como parámetros la dirección de la matriz y la dirección del arreglo de estructuras, procederá a determinar las direcciones del valor máximo y del valor mínimo de cada fila de la matriz.

Estas direcciones quedarán almacenadas en los correspondientes miembros **Max** y **Min**. La búsqueda de estos extremos se realizará mediante los miembros mencionados.

Una tercera función llamada **MostrarDatos( )** visualizará por pantalla y en forma matricial, los valores aleatorios generados.

Una cuarta función denominada **MostrarProcesamientos( )** visualizará por pantalla la dirección de comienzo de cada fila de la matriz y la posición de **Max** y **Min** dentro de dicha fila.

NOTA
Las dimensiones de los arreglos conviene tomarlas como constantes DIM1, DIM2, etc. de manera que si realizamos modificaciones todo se ajuste automáticamente.

```
//--------------------------------EjeClase4_11.cpp
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <process.h>

typedef struct  {
                  float *Max;
                  float *Min;
                }TExtr;
const int DIM1 = 5;
const int DIM2 = 5;

void   *ReservarMemoria        ( int Total               );
void    ProcesarDatos          ( float **p1, TExtr *p2   );
void    MostrarDatos           ( float **p               );
void    MostrarProcesamientos  ( float **p1, TExtr *p2   );

// ------------------------------------------------------------



void main()
{
  float    **p1;
  TExtr     *p2;
  int       i,j;

  clrscr(); highvideo(); randomize();

  // --- RESERVA PARA LAS ESTRUCTURAS DE PROCESAMIENTO -----
  p2=(TExtr *)ReservarMemoria(DIM1*sizeof(TExtr));

  // --- RESERVAS PARA LA MATRIZ DINAMICA -----------------
  p1=(float **)ReservarMemoria(DIM1*sizeof(float *));
  for(i=0;i<DIM1;i++) {
      p1[i]=(float *)ReservarMemoria(DIM2*sizeof(float));
```

```
      for(j=0;j<DIM2;j++)
        p1[i][j]=(5000+random(5001))*0.01;
  }

  ProcesarDatos(p1,p2);
  MostrarDatos(p1);
  MostrarProcesamientos(p1,p2);

  getch();
}
// ------------------------------------------------------------
void *ReservarMemoria(int Total)
{
  void *p;
  if((p=malloc(Total))==NULL) {
      cprintf("NO PUDO RESERVAR MEMORIA DINAMICA");
      getch(); exit(1);
  }
  return(p);
}
// ------------------------------------------------------------
void ProcesarDatos(float **p1, TExtr *p2)
{
  int      i,j;
  float    Max;
  float    Min;
  float   *DirMax;
  float   *DirMin;

  for(i=0;i<DIM1;i++) {
      Max=0.0;  Min=999.0;
      // --- INICIALIZA DIRECCIONES DE MIEMBROS ----
      p2[i].Max=&(p1[i][0]);
      p2[i].Min=&(p1[i][0]);

      for(j=0;j<DIM2;j++) {
             if(((p2[i].Max)[j]) > Max) {
                  Max=(p2[i].Max)[j];
                  DirMax=&((p2[i].Max)[j]);
             }
             if(((p2[i].Min)[j]) < Min) {
                  Min=(p2[i].Min)[j];
                  DirMin=&((p2[i].Min)[j]);
             }
      }

      p2[i].Max=DirMax;
      p2[i].Min=DirMin;
  }
}

// ------------------------------------------------------------
void MostrarDatos(float **p)
{
```

```
  int i,j;

  textcolor(WHITE);
  cprintf(" --- DATOS ALMACENADOS EN LA MATRIZ DINAMICA ---
\r\n");
  cprintf("\r\n");

  textcolor(LIGHTRED);
  for(i=0;i<DIM1;i++) {
      for(j=0;j<DIM2;j++) cprintf("%6.2f",p[i][j]);
      cprintf("\r\n");
  }
  cprintf("\r\n");
}
// ------------------------------------------------------------
void MostrarProcesamientos(float **p1, TExtr *p2)
{
  int i,j;

  textcolor(WHITE);
  cprintf(" --- POSICIONES DE MAXIMOS Y MINIMOS Y SUS VALORES --
-\r\n\r\n");
  textcolor(LIGHTGREEN);

  for(i=0;i<DIM1;i++) {
      cprintf(" Direcc_Fila %d = %u\r\n" ,i,&p1[i][0]);
      cprintf("       Direcc_Max       =       %u,       valor:
%6.2f",p2[i].Max,*p2[i].Max);
      cprintf("       Direcc_Min       =       %u,       valor:
%6.2f",p2[i].Min,*p2[i].Min);
      cprintf("\r\n");
  }
}
// ------------------------------------------------------------
```