UNIVERSIDAD NACIONAL
DE
SAN MARTÍN

Tecnicatura Universitaria en Programación Informática
y
Tecnicatura Universitaria en Redes Informáticas

Laboratorio de Computación II

MÓDULO IV

Mg. Lic. Mónica Hencek
Lic. Juan José López

Vectores (arrays)

Un vector es un conjunto de datos del mismo tipo, ordenados 'secuencialmente' y almacenados de forma contigua (es decir uno al lado del otro) en memoria principal. Cada elemento se puede manipular usando el nombre del vector y la posicion del elemento. Se puede trabajar con vectores de enteros, de reales, de caracteres, con vectores de vectores, etc. Las estructuras con las que trabajaremos a partir de ahora pueden 'anidarse' y aumentar su complejidad tanto como sea necesario. De aquí en más, al hablar de 'vector' o 'array', nos referiremos a los vectores o arrays unidimensionales.

Al declarar un vector se indica el tipo de dato que almacena, el nombre del vector y el tamano del mismo, es decir, la cantidad de elementos que contiene.

Ejemplo: Declaración de un vector de 5 enteros:

int v[4];

La cantidad de elementos del vector se indica entre corchetes. El primer elemento del array se indica como v[0], el segundo es v[1], el tercero es v[2] el cuarto es v[3], y así sucesivamente. El elemento que determina una posicion en le vector se llama subíndice. Los subíndices en C son enteros, y el primer subíndice siempre es 0. Para asignar valores a los elementos del vector se puede hacer:

v[0]=2;

v[1]=3;

Tambien valen sentencias como

v[2]=v[0]+v[1];

O podemos ingresar valores desde teclado:

```
printf("Ingrese valor para v[3]\n");
scanf("%d", &v[3]);
```

Podemos ver el contenido del vector usando un ciclo:

```
int i;
for(i=0;i<4;i=i+1)
printf("El valor del elemento v[%d] es %d\n",i,v[i]);
```

Es frecuente ingresar todos los valores de um array desde teclado, haciendo un recorrido ‘por filas’ o ‘por columnas’. En este caso, se recorre por filas:

```
for(i=0;i<4;i=i+1)
{ printf("Ingrese valor para v[%d]\n",i);
  scanf("%d", &v[i]);}
```

Si nos interesara calcular la sumatoria de los valores del vector, podriamos hacer:

```
int suma=0;
for(i=0;i<4;i=i+1)
suma=suma+v[i];
```

y luego emitimos el resultado:

```
printf("La sumatoria de los elementos del vector es %d\n", suma);
```

Si queremos ordenar el vector de menor a mayor, una forma de hacerla es la podemos hacer lo siguiente:

- Comparamos el primer elemento del vector con el segundo, y si 'estan desordenados' los intercambiamos;
- Luego comparamos primer elemento y tercero, y si es necesario, intercambiamos.
- Luego hacemos lo mismo considerando el primer elemento y el cuarto, y así sucesivamente hasta conseguir que el elemento más pequeño este ubicado en el primer lugar.
- Entonces procedemos a hacer lo mismo comparando y eventualmente intercambiando el segundo elemento con el tercero, luego el segundo con el cuarto, etc, hasta que el valor correcto quede almacenado en el segundo lugar.
- Así repetimos hasta que el array quede ordenado.

Ejemplo:

Para ordenar esta secuencia por el algoritmo de **ordenamiento por seleccion** de forma creciente,

9 7 4 6

Comparamos 9 con 7. Como estan desordenados, los intercambiamos. Queda

7 9 4 6

Comparamos 7 con 4. Como estan desordenados, los intercambiamos. Queda

4 9 7 6

Comparamos 4 con 6. Como no estan desordenados, quedan igual. Ahora tenemos al menor elemento en la primera posicion. Repetimos el mismo esquema, pero en la ‘zona

desordenada' que es la que va de la posicion 1 hasta la 3. De este modo, se compara 9 con 7 y se intercambian

4 7 9 6

Se compara 7 con 6 y se intercambian

4 6 9 7

ahora el elemento de la posicion 1 es correcto. Repetimos el esquema en la zona desordenada, que va desde la posicion 2 hasta la 3. Se compara 9 con 7 y se intercambia

4 6 7 9

El array esta ordenado. A continuación tenemos el código de ese ordenamiento, que es el ordenamiento 'por seleccion' y corresponde a uno de los mas sencillos algoritmos para ordenar un vector. La variable n almacenará la cantidad de elementos del vector, las variable i y j, corresponden a los subíndices de los elementos que se comparan, aux es una variable auxiliar para realizar el intercambio de valores, cuando corresponda:

```
int n=4,i ,j ,aux;
for(i=0; i<n-1; i++ )
   for(j=i+1; j<n; j++)
      if (v[i]>v[j])
      {  aux=v[i];
         v[i]=v[j];
         v[j]=aux; }
```

Ahora mostramos el array ordenado:

```
printf("Vector ordenado\n");
for(i=0;i<4;i=i+1)
printf("El valor del elemento v[%d] es %d\n",i,v[i]);
```

Relación entre los arrays y los punteros:

En C hay una relación intrínseca que liga un array de la zona de memoria 'estática' (segmento de datos) con la dirección en donde este se encuentre. Consideremos un array:

int v[5];

al que ya hemos cargado con datos.

En C , **el nombre de un array es la dirección de comienzo del mismo.** Es decir que se verifica:

v es &v[0]

v es un *Puntero Constante*, es decir, *no podemos cambiar su valor porque eso equivale a que el array comience en otra parte, y los datos del segmento de datos 'no se mueven'.*

Si v es &v[o]

Entonces v+1 es &v[1]

v+2 es &v[2]

y así sucesivamente. Lo cual implica, por la notación de punteros que ya conocemos, que

v[0] es *v

v[1] es *(v+1)

v[2] es *(v+2)

Si queremos emitir el contenido del array v usando la notación de punteros, se puede hacer:

```
printf("Emision del vector usando notacion de punteros y el nombre del vector\n") ;
for(i=0;i<5;i++)
printf("v[%d]=%d \n", i, *(v+i)) ;
```

Arrays y strings en C

Un string es una cadena de caracteres, que nos permite, por ejemplo , manipular una palabra. En C un string se maneja con un vector de caracteres que tiene un caracter al final indicando que ahi termina la misma. Ese terminador se indica con **\0**, y corresponde al ASCII 0. Ejemplo: Si usamos un array para almacenar una palabra, por ejemplo HOLA en el array palabra se almacena así: `char palabra[10];`

podemos hacer

```
palabra[0]= 'H';
palabra[1]= 'O';
palabra[2]= 'L';
palabra[3]= 'A';
```

Ahora ponemos el terminador (el caracter \0, como dijimos antes)

```
palabra[4]= '\0' ;
```

Las posiciones no usadas del array no interesan, solo debe quedar el terminador a continuacion de la última letra de la palabra. Para mostrar la palabra, se puede usar un ciclo (lo cual no es muy frecuente), como se muestra a continuacion:

```
int i=0;
printf("\n emision de la palabra caracter a caracter con el especificador c \n");
while(palabra[i]!='\0')
{
printf("%c", palabra[i]);
i++;
}
```

O bien podemos usar el especificador de formato %s (mucho más usual), que se usa para entrada y salida de cadenas de caracteres. Entonces:

```
printf("\n emision de la palabra usando el especificador s \n");
printf("%s", palabra);
```

Observa que, al indicar palabra, estamos dando la dirección de comienzo de la misma como sucede con los nombres de los vectores en general. Se puede ingresar una palabra caracter a caracter, como se indica a continuación (en este caso, el usuario debe pulsar enter cuando haya terminado de ingresar los caracteres):

```
printf("\n ingreso de la palabra letra a letra\n");
for(i=0;i<10; i++)
 { printf("\n Ingrese letra \n");
 fflush(stdin);
 scanf("%c", &palabra[i]); }
```

También se puede ingresar la palabra de una sola vez , y no letra a letra, con un scanf y el especificador %s, de esta forma:

```
printf("\n ingreso de la palabra con scanf y %\n");
printf("\n ingrese la palabra y pulse enter al finalizar\n");
scanf("%s", palabra);
```

Una cuestión importante: palabra en el scanf no lleva & porque “palabra” es la DIRECCIÓN de comienzo vector, palabra es un PUNTERO al comienzo de la palabra.

Entonces scanf ‘ya tiene’ la dirección en donde debe almacenar la secuencia de caracteres. Cuando scanf detecta el caracter correspondiente al enter, almacena en esa posicion y en las siguientes (si sobrara espacio en el array) un \0.

La biblioteca string.h

C provee en esta biblioteca, muchas funcione sutiles para realizar las operaciones basicas con strings, tales como comparación, asignación, etc.
Investiga y ejemplifica el uso de strcmp y strcpy.

Arrays y funciones:

Dado que el nombre de un vector ES su dirección de comienzo, cuando se pasa un vector a una función siempre se esta pasando la dirección de comienzo del mismo.
El vector siempre pasa 'por puntero' a las funciones, ***ya sea que estas alteren o no el contenido del vector***. En algunos libros se puede encontrar la expresion 'un vector siempre pasa por referencia', nosotros preferimos no usarla, porque el concepto de referencia es muy preciso en C++, y no corresponde a un puntero.

Ejemplos de funciones que trabajan sobre arrays pasados como argumento

Consideremos una constante
const int m=10;
y un array
float v[m];

Función para cargar datos en un array de float: Recibe como argumento el array y un entero correspondiente al tamaño del mismo La notación [], indica que se recibe un vector pero se omite el tamaño del mismo; ya que entre los corchetes no hay valor; este tamaño se indica con el segundo parámetro:

```
void CargaVec (float w[], int s )
 {  int i; //variable auxiliar para el ciclo de lectura
  for(i=0;i<s;i++)
```

```
{ printf("\n Ingrese el valor del elemento de posicion %d  ", i);
  scanf("%f", &w[i]); }
}
```

Función para ordenar un vector de float

```
void OrdenaVec (float w[], int s)
 {
   int i,j;
   float aux; //variables auxiliares  para el
ordenamiento por seleccion
   for(i=0;i<s-1;i++)
      for(j=i+1; j<s;j++)
        if (w[i]>w[j])
          { aux=w[i];
            w[i]=w[j];
            w[j]=aux; }
      }
```

Función para determinar el valor del mayor elemento de un array de float. (Retorna el valor de ese máximo).

```
float MayorEleVec (float w[], int s )
{ int i;
 float may=w[0];
 for(i=1; i<s ;i++)
       if (w[i]>may)
          may=w[i];
 return may;  }
```

Función que emite un vector de float

```
void EmiteVec(float w[],int s)
{ int i;
 for(i=0;i<s;i++)
  printf("%f\n", w[i]); }
```

En cada caso, la función recibe al array y el tamaño del mismo; esto permite que la función en cuestión sea reutilizada com arrays de distinto tamaño; main podria ser así (aquí veremos las invocaciones respectivas). Consideremos que queremos ingresar un array de reales, ordenarlo y emitirlo.

```
int main()
{const int m=10;
float v[m];
```

```
float h;
CargaVec(v,m);
h= MayorEleVec (v], m);
printf("El mayor elemento del array es %f", h);
OrdenaVec(v,m);
EmiteVec(v,m);
system("pause");
return 0;}
```

Arrays multidimensionales:

Un array es una estructura de datos que contiene datos de un mismo tipo, organizados de un modo particular. Si el array es de una dimension, como los que hemos visto antes, decimos que los datos, que son del mismo tipo, estan 'organizados secuencialmente'. Esta idea se generaliza a otras configuraciones posibles, y por eso hablamos de arrays bidimensionales, tridimensionales, y más. De este modo, es posible tratar un conjunto de datos del mismo tipo como si estuvieran dispuestos en una grilla (dos dimensiones) o apilados por capas en tres dimensiones, o en más. Un array bidimensional o 'matriz' es una abstracción del lenguaje que nos permite tratar a esos datos como en una tabla. El array tiene un nombre, y para ubicar un dato en particular es necesario indicar la fila y la columna correspondiente. Tanto las filas como las columnas se numeran comenzando en 0. Ejemplo:

3	5	-8
9	0	2
34	-8	-1
4	0	5

Este array tiene 4 filas y 3 columnas. Si lo llamamos D, es:

D[0][0] vale 3
D[0][2] vale -8
D[1][1] vale 0
D[3][4] vale 5
etc...

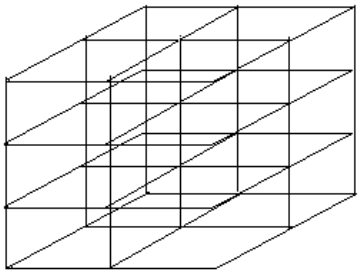
La definicion de este array puede ser:

```
int D[4][3];
```

Mediante dos pares de corchetes indicamos que se requieren dos subíndices para referirnos a un dato contenido en el array. Análogamente, la definición:

```
int H [2][4][3];
```

corresponde a un array tridimensional, por ejemplo:



Utilizando funciones para manipular una matriz de enteros

Dada este array bidimensional de 4 filas y tres columnas,

```
int mat[4][3];
```

La carga de esta matriz puede hacerse desde teclado, asi; (usamos variables auxiliares i y j para recorrerla):

```
int i, j;
for (i=0;i<4;i++)
    for (j=0;j<3;j++)
        {printf("\n Ingrese el valor de mat[%d][%d]: ", i,j);
         scanf("%d", &mat[i][j]);
         }
```

Para emitir la matriz:

```
printf("\nContenido de la matriz mat\n");
for (i=0;i<4;i++)
    for (j=0;j<3;j++)
          printf("\n mat[%d][%d]:  %d\n", i,j,mat[i][j] );
```

Si quisieramos recorrer la matriz ‘por columnas’, se podría hacer:

```
printf("\nRecorrido de la matriz mat por columnas:\n");
for(i=0;i<3;i++)
  for(j=0;j<4;j++)
      printf("\n mat[%d][%d]:  %d\n", i,j,mat[j][i] );
```

Matrices y punteros

La relación que liga los arrays y los punteros, que hemos visto al tratar los arrays unidimensionales, en el caso de los arrays bidimensionales, indica que, dada una matriz denominada mat, por ejemplo:

int mat[4][3];

se verifica que mat corresponde a la dirección de comienzo de la misma. Es decir:

mat es &mat[0][0]

En C, una matriz se maneja como un array de arrays. Eso hace, que sea válido usar el nombre de la matriz con un solo subíndice para indicar la dirección de comienzo de cada fila. Asi,

mat identifica el array de direcciones de filas.

mat es mat[0]

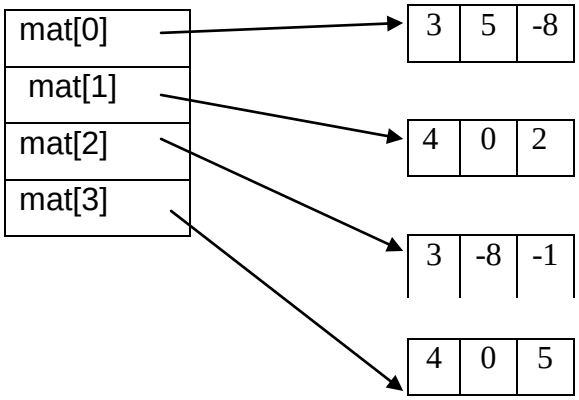
mat es &mat[0][0]

mat +1 es mat[1]

mat[1] es &mat[1][0]

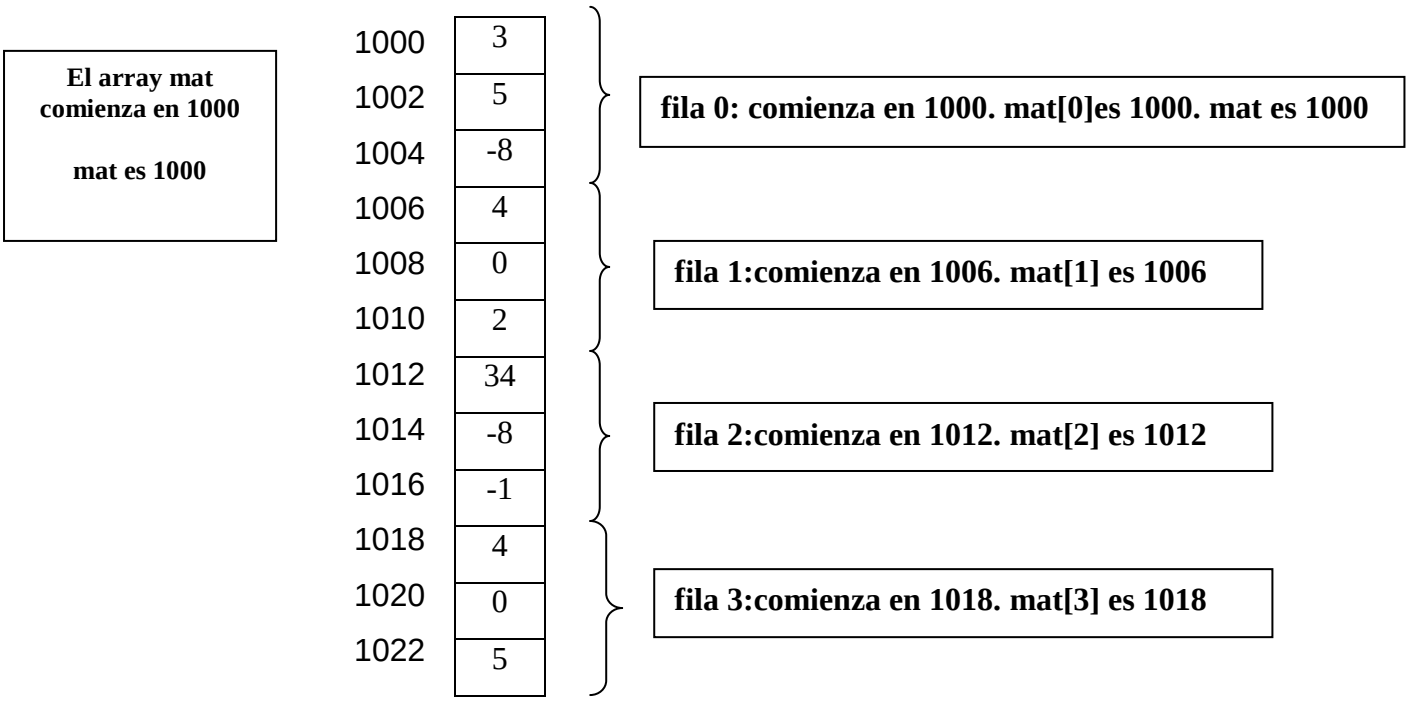
La direccion de cada fila i es mat[i] , con un solo subindice

(y la direccion de un elemento se puede obtener como siempre, con &mat[i][j]). Podemos interpretar lo que el C nos ofrece para trabajar con la matriz mat del siguiente modo:



mat = mat[0]=&mat[0][0]

Si representamos los datos de la matriz tal como se almacenan, se puede hacer esta representación:(vamos a considerar que la dirección de comienzo del array es 1000 y que cada dato usa 2 bytes). En esta representación, dibujamos un casillero por cada dato del array.(recordá que en realidad, cada entero utiliza al menos 2 bytes o celdas).



si mat[2] es la direccion de la fila 3, entonces, considerando el operador de desreferencia *, se observa que

mat[2][0] es * mat[2]
mat[2][1] es * (mat[2]+1)
mat[2][2] es * (mat[2]+2), etc,

Ciclo de ejemplo:

```
printf("\n Contenido de la matriz :\n");
for(i=0;i<4;i++)
 for (j=0;j<3;j++)  printf("\n mat[i][j] : %d\n", *(mat[i]+j));
```

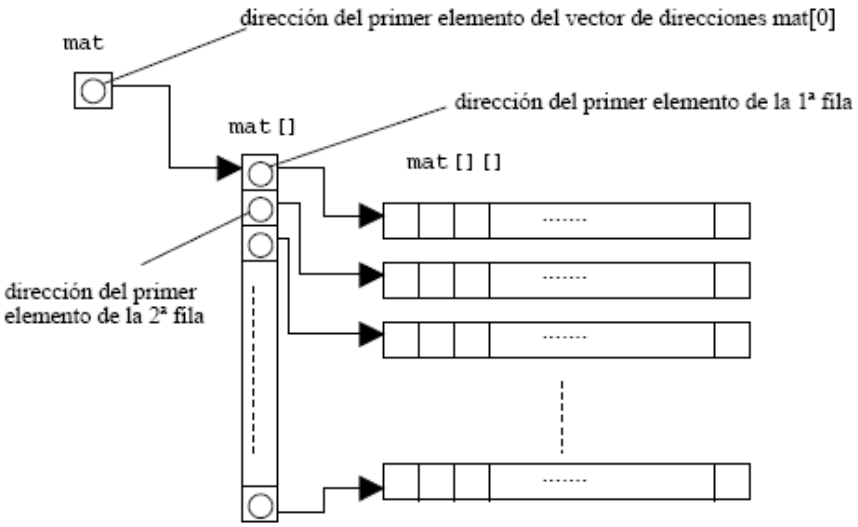
Siguiendo el mismo razonamiento, si mat es la dirección de comienzo del array de direcciones de filas, considerando una fila en particular, por ejemplo 3:

mat[2][0] es *(* mat+2)
mat[2][1] es *((* mat+2)+1)
mat[2][2] es *((* mat+2)+2), etc,

Ciclo de ejemplo:

```
printf("\n Contenido de la matriz :\n");
for(i=0;i<4;i++)
  for (j=0;j<3;j++)
    printf("\n mat[i][j] : %d\n", *(*(mat+i)+j));
```

La figura siguiente resume gráficamente la relación entre matrices y vectores de punteros.



Registros o struct

Un registro o struct, como se lo llama en C, puede definirse como array de elemento heterogéneos. Dentro de un struct puede haber elementos de distinto tipo. Cada elemento se referencia mediante su nombre ‘de campo’. En el siguiente se mostrará una struct que contiene un legajo, un curso (char) y una nota, cada 'parte' del struct se llama campo y tiene un identificador. Ten en cuenta también que lo que estamos declarando es un tipo de dato, no se genera con estas líneas espacio de memoria para contener variables sino que **se esta dando la descripción del tipo**.

```
struct alumno
{ int legajo;
 char curso;
 float nota; };
```

Así se declaran dos variables, a y b, de este tipo:

```
alumno a, b;
```

Cada campo del struct se referencia mediante el nombre de la variable, seguido de un punto (operador selector de campo) y el nombre del campo correspondiente. Entonces, si la variable struct se llama a, entonces

a.legajo es el campo legajo de a
a.curso es el campo curso de a
a.nota es el campo nota de a

Ejemplo:

```
a.legajo=100;
a.curso='c' ;
```

O, usando scanf

```
printf("ingrese la nota ");
scanf("%f", &a.nota);
```

Ten en cuenta que si el campo legajo es un entero int, todo lo que se hace con un int se puede hacer con a.legajo y con b.legajo. Todo lo que se hace con un char se puede hacer con a.curso y b.curso. Todo lo que se haga con un float se puede hacer con a.nota y con b.nota. Ejemplo de salida por pantalla:

```
printf("La variable a contiene\n");
printf("a.legajo = %d\n", a.legajo);
printf("a.curso = %c\n", a.curso);
printf("a.nota = %f\n", a.nota);
printf("La variable b contiene\n");
printf("b.legajo = %d\n", b.legajo);
printf("b.curso = %c\n", b.curso);
printf("b.nota = %f\n", b.nota);
```

Punteros y struct:

Para declarar un puntero a la struct alumno, la sintaxis es:

```
alumno *p;
```

Para manipular las variables a y b mediante el puntero p se puede usar el operador -> que permite seleccionar un campo de una variable struct. Es decir, que si p apunta a a

```
p=&a;
```

Se puede hacer, para emitir a

```
printf("\nEl legajo de la struct a es %d\n", p->legajo);
printf("\nEl curso de la struct a es %d\n", p->curso);
printf("\nLa nota de la struct a es %d\n", p->nota);
```

Luego se puede hacer

```
p=&b;
```

y emitir el contenido del otro struct con

```
printf("\nEl legajo de la struct b es %d\n", p->legajo);
```

```
printf("\nEl curso de la struct b es %d\n", p->curso);
printf("\nLa nota de la struct b es %d\n", p->nota);
```

También se puede usar el operador indirección * para manipular el contenido del struct

p->legajo tambien es *(p.legajo)

Usando esa notación, se puede hacer:

```
printf("\nEl legajo de la struct b es %d\n", (*p).legajo);
printf("\nEl curso de la struct b es %d\n", (*p)curso);
printf("\nLa nota de la struct b es %d\n", (*p)nota);
```

Ejemplo integrador de manejo de funciones, punteros, arrays y structs:

/*Es te programa utiliza un array de N registros cada registro es una struct alumno. Mediante un menú se permite: 1: vaciar el array 2: agregar un muevo registro, si la capacidad lo permite 3: borrar un registro indicado por su legajo, si este existe 4: listar el contenido del array 5: salir del programa. Además, el array de registros siempre mantendrá los registros ordenados por legajo. Es decir, que organizaremos la información como una lista ordenada de registros. En este caso, al organizar el programa colocaremos antes de main los prototipos de las funciones utilizadas, y luego de main, las definiciones de las funciones usadas. El programa esta totalmente modularizado */

```
#include<stdio.h>
#include<stdlib.h>
        struct alumno
        {  int legajo;
         char curso;
         float nota;   };
  /*Emitirmensaje despliega las opciones del menu*/
        void Emitirmensaje(); /*Menu llama a Emitirmensaje y luego pide la opción*/
        int Menu();
//Vaciar deja el array  con todas las posiciones disponibles
        void Vaciar(int *q);
//Leer ingresa los datos del curso y nota de un struct desde teclado
        void Leercursonota(alumno *);
//Leerleg ingresa un legajo desde teclado
```

```
        void Leerleg(int *);
//Esta en retorna el subíndice en el cual esta el legajo que se pasa, o -1 si no esta
        int Estaen(int, alumno [], int );
//Posicion retorna la posición que le corresponde al nuevo registro en el array ordenado
        int Posicion(int , alumno [], int );
//Correr_der desplaza elementos a derecha para hacer lugar para una inserción
        void Correr_der (alumno [], int, int);
//Correr_izq desplaza elementos a izquierdz para borrar un registro 'por solapamiento'
        void Correr_izq(alumno [], int , int);
//Ubicar coloca el nuevo registro en la posición indicada
        void Ubicar (alumno , alumno [], int );
//Mensaje que indica que el legajo ya estaba
        void Error1();
//Mensaje que indica que el array esta lleno
        void Error2();
//Mensaje que indica que el array esta vacio
        void Error3();
//Mensaje que indica que el legajo no esta en el array
        void Error4();
//Agregar determina la posición de inserción, corre elementos para hacer espacio,
//ubica el registro en posicion y incrementa la ultima posición
        void Agregar(alumno , alumno [], int *);
//Insertar recibe el array de registros, ordena leer el nuevo registro y llama a Agregar
        void Insertar(alumno [], int , int *);
//Borrar recibe el array de registros, ordena leer el legajo del registro que debe borrarse,
//ubica la posición del registro y borra por corrimiento, encimando los registros
//y decrementando la ultima posición
        void Borrar(alumno [],int , int *);
//Emitealu enite el contenido de un registro
        void Emitealu (struct alumno );
//Listar lista el contenido del vector de registros
        void Listar(alumno [], int);
//Decrementar disminuye en 1 la posicion del ultimo elemento
        void Decrementar(int *);
```

```
//Incrementar aumenta en 1 la posicion del ultimo elemento
                void Incrementar(int *);
```

```
main()
{
const int N=10;
int ult=-1;     //la variable ult almacena la ultima posición ocupada del array
                // se usará para verificar si hay o no espacio en el array
alumno v[N];
int opcion;
opcion = Menu();
while(opcion!=5)
{
   switch(opcion)
   {
   case 1: Vaciar(&ult); //se vacia el vector poniendo el ultino escrito en -1
           break;
   case 2: Insertar (v,N,&ult);
   //si hay lugar, y se realiza la inserción, hay que incrementar ult, por eso se pasa su dirección
           break;
   case 3: Borrar(v,N, &ult);
    //si hay lugar, y se realiza el borrado, hay que decrementar ult, por eso se pasa su dirección
           break;
   case 4: Listar(v,ult) ;
           break;
    }
     opcion=Menu(); //volvemos a llamar a Menu()
}
system ("pause");
return 0;  }
```

```
void Emitirmensaje()
```

```
{
printf("\nMenu de opciones\n");
printf("\n 1: Borrar el contenido del vector de registros\n");
printf("\n 2: Agregar un registro nuevo al vector de registros\n");
printf("\n 3: Borrar un elemento del vector de registros\n");
printf("\n 4: Listar el contenido del vector de registros\n");
printf("\n 5: Salir del programa\n");
printf("\nIngrese opcion\n");
}
```

```
int Menu()
{
 int aux;
 Emitirmensaje();
 scanf("%d", &aux) ;
 while((aux<1) || (aux>5))
    {
      printf("ERROR de OPCION, ingrese nuevamente\n") ;
       scanf("%d", &aux) ;
    }
 return aux;
    }
```

```
void Leercursonota(alumno *a)
{
  fflush(stdin);
  printf("\nIngrese el curso\n") ;
  scanf("%c", &(a->curso)) ;
  fflush(stdin);
  printf("\nIngrese la nota\n") ;
  scanf("%f", &(a->nota)) ;   }
```

```
int Estaen(int l, alumno w[], int ultimo)
```

```
{
int i=0,posi=-1;
int encontrado=0;
while ((i <= ultimo)&&(encontrado==0))
      if (w[i].legajo==l)
                  {
                     posi=i;
                     encontrado=1;
                  }
         else i++;
return posi; }
```

```
int Posicion(int l, alumno w[], int ultimo)
{
 int j=0;   //variable auxiliar para el recorrido
 int posi, encontrado=0;
 while((j<=ultimo) && (encontrado==0))
 {
  if (l< w[j].legajo)
             {
                  posi=j;  //esta es la posición para la inserción
                  encontrado=1;
             }
     else j++;
 }
 if (encontrado ==0) posi=ultimo +1;  //si no se encontró antes la posición, el nuevo registro sigue al ultimo
 return  posi; }
```

```
void Correr_der (alumno w[], int posi, int ult)
{
 int j = ult;
 for (;j>=posi ;j--)
```

```
    w[j+1]=w[j];        //corre a derecha, comenzando en la ultima posición cargada, para hacer espacio en posi }
```

```
void Ubicar (alumno alu, alumno w[], int posi)
{  w[posi]=alu;   //carga el registro en la posicion indicada }
```

```
 void Incrementar(int *q)
 {  (*q)++;  }
```

```
 void Error1()
 {   printf ("\nEse legajo ya estaba almacenado; imposible hacer el alta\n") ;    }
```

```
 void Error2()
 {   printf ("\nEl array esta lleno; imposible hacer el alta\n") ;    }
```

```
void Agregar(alumno alu, alumno w[], int * pult)
{
 int pos = Posicion(alu.legajo, w, *pult) ; //Esta función retorna la posición que
                                  // le corresponde a alu, según su legajo, en el vector w
 Correr_der(w,pos, *pult);     //Esta función 'corre' los elementos del array w, desde la posición
 //indicada por pos, hasta la ultima, para hacer lugar al nuevo elemento
 Ubicar(alu, w, pos);
 Incrementar(pult) ; //se incrementa la variable que contiene la posición
                  //del ultimo elemento del array, usando un puntero}
```

```
void Insertar(alumno w[], int N, int *p)
{
 struct alumno aux;    //auxiliar para la lectura
 if (*p < N-1)         //chequeo si hay espacio disponible
    {
      Leerleg (&(aux.legajo));     //se carga aux; para eso se pasa su direccion
```

```
    if ((Estaen(aux.legajo, w, *p))!= -1)     // si Esta retorna -1 ese legajo no estaba previamente
        Error1();   //La funcion Error1 emite un mensaje indicando que el legajo estaba
      else
        {  Leercursonota (&aux);//lee los campos que faltan
        Agregar(aux, w, p)  ;  }
       }
 else Error2() ;     //la funcion Error2 emite un mensaje indicando que el array esta lleno   }
```

```
void Emitealu (struct alumno a)
{  printf("\nLegajo: %d\n", a.legajo);
  printf("Curso: %c\n", a.curso);
  printf("Nota: %f\n", a.nota);  }
```

```
void Listar(alumno w[], int u)
{  int i;
 for(i=0;i<=u;i++)     //se emite el contenido del array de registros
    Emitealu (w[i]); }
```

```
void Vaciar(int *q)
{ *q=-1;  }
```

```
void Error3()
{ printf("\nImposible borrar, el array esta vacío\n");  }
```

```
void Error4()
{printf("\nImposible borrar, ese legajo no esta en el array\n");      }
```

```
void Leerleg(int *pl)
{  printf("\nIngrese el legajo\n");
 scanf("%d",pl);   }
```

```
void Correr_izq(alumno w[], int posi, int ult)
```

```
{ int i;
for (i=posi;i<=ult-1;i++)
    w[i]=w[i+1]; }
```

```
void Decrementar(int *q)
{ (*q)--;  }
```

```
void Borrar(alumno v[],int N, int *pult)
{
  if (*pult==-1) Error3();  // array vacio, mensaje de error
    else
    {
     int leg;
     Leerleg(&leg); //se lee un legajo
     int i=Estaen(leg, v, *pult);  //se ubica la posicion
     if (i!=-1)       //si i ==-1 el legajo no esta
              { Correr_izq (v, i,*pult); //corre, encima y 'borra registros'
                Decrementar (pult);    //decrementa la ultima posición  }
        else Error4();  //el legajo no esta, mensaje de error
     }}
```