# ARREGLOS

## INTRODUCCIÓN

En este capitulo vamos a introducirnos en el importante tema del manejo de las estructuras de datos. Frecuentemente como diseñadores de software se van a encontrar ante el problema de tener que manejar conjuntos de datos relacionados entre si, de diferentes Tipos, comúnmente llamados registros ,por ejemplo el caso de una agenda donde tenemos cada registro con : nro teléfono , nombre , apellido, sexo, dirección , DNI etc…Todos datos asociados a una persona y lo solucionaremos mediante el uso de estructuras, denominadas **"struct"** en Lenguaje C (que veremos detalladamente en los próximos capítulos) o conjuntos de datos que son del mismo Tipo por ejemplo las notas de todas las materias que están cursando agrupadas en un arreglo o vector (denominado **"array"** en Lenguaje C) en el cual no varia su tamaño o sea pueden guardar una cantidad definida a priori de datos. Como no se puede modificar su tamaño en tiempo de ejecución, son denominadas **estructuras estáticas** una de ellas "**El arreglo**", es el motivo de nuestro estudio en este capitulo. En un posterior Curso mas avanzado conocerán otros tipos de estructuras como son las dinámicas, Listas, Colas, Pilas y Árboles que pueden variar su tamaño en tiempo de ejecución.

## ¿QUÉ ES UN ARREGLO?

Una primera asociación o relación que podemos hacer con conocimientos previos es pensar un arreglo como un vector o como una matriz. (Conocidos a través del álgebra) En este caso el arreglo unidimencional posee un nombre o identificador, al igual que un vector y también un subíndice que indica la posiciòn a la cual queremos acceder, de todas las que componen el vector. Solo existen algunas diferencias en cuanto a la nomenclatura. Otra asociación matemática, haciendo uso del concepto de conjunto, es la siguiente: podemos pensar un arreglo como un conjunto de elementos que tienen una propiedad en común en este caso es el tipo de dato y el nombre del conjunto y podemos acceder a estos elementos a través de un nombre y un subíndice es mas nosotros podremos inicializar un arreglo asignándoles los datos con la misma nomenclatura que usamos en conjuntos, solo que en este caso se debe tener en cuenta el orden . A continuación vamos a ver detalladamente el asunto.

Conocer acabadamente nuestra primera estructura de datos no es tan complicado como parece .Hasta el momento nosotros conocemos como almacenar un dato en una variable si bien pueden ser de distintos tipos, un entero, un carácter, un numero de punto flotante, es un único dato el que podemos guardar. Cuando necesitábamos procesar un conjunto de datos recurríamos a las estructuras repetitivas y a ingresarlas consecutivamente pero no podíamos almacenar toda la información procesada y relacionada. Contando con estructuras como los arreglos podemos solucionar este problema. Un arreglo es un conjunto de posiciones de memoria relacionadas a través de un nombre o identificador común y la limitación que tiene esta estructura es que los datos que guardan esas posiciones de memoria tienen que ser del mismo tipo y la cantidad de datos que queremos guardar debe estar definida. Para acceder a cualquiera de estos datos o elementos basta dar el nombre del arreglo seguido de la

posición numérica entre corchetes del dato almacenado. Siguiendo el ejemplo de la figura 1 que se encuentra a continuación el primer elemento del arreglo se conoce como Arrayej[0] ,el segundo Arrayej[1],el tercero Arrayej[2] y así sucesivamente hasta el ultimo el noveno en orden Arrayej[8].

Figura1

El identificador del arreglo es Arrayej y la posición es el entero que se encuentra entre corchetes y varia entre

0 y Tam –1 en este caso Tam es = 9

| | |
|---|---|
| 'a' | Arrayej[0] |
| 'b' | Arrayej[1] |
| 'c' | Arrayej[2] |
| 'd' | Arrayej[3] |
| 'e' | Arrayej[4] |
| 'f' | Arrayej[5] |
| 'g' | Arrayej[6] |
| 'h' | Arrayej[7] |
| 'i' | Arrayej[8] |

Lo primero que tenemos que conocer para trabajar con arreglos al igual que con el tema de variables es como se declaran y se inicializan arreglos .

Forma general de declaración de arreglos en "C" :

**Tipo IdentifArreglo1[Tamañoarreglo1] ,..., IdentifArreglon [Tamañoarreglo n ];**

Ejemplo: en el caso de la figura 1 la declaración seria así:

**char Array[9];** En este caso estaría únicamente declarando El arreglo o sea El compilador reserva 9 elementos o posiciones de memoria de un byte para luego en otra parte del programa asignarle cada uno de los caracteres.

**char Array[ ]={a,b,c,d,e,f,g,h,i,j};** En este otro caso el compilador determina la cantidad de datos o inicializadores de los que se encuentran listados entre llaves en este caso nueve, de esta manera el arreglo es declarado e inicializado al mismo tiempo

Si trabajamos con datos del tipo entero y quisiéramos declarar el arreglo e inicializarlo a cero seria de esta manera:

**int arraynotas[9 ]={ 0 };** En este otro caso el compilador determina que tiene que reservar nueve elementos debido al numero que se encuentra entre corchetes , pero la cantidad de inicializadores es uno , en este caso el compilador carga con cero las demás elementos del arreglo

## EJEMPLO CON ARREGLOS UNIDIMENCIONALES (VECTORES)

A continuación vamos a escribir un programa que cargue Array[ 9 ] con caracteres y Arraynotas[ 9 ] con enteros y Luego muestre sus contenidos y las direcciones de memoria donde se encuentran almacenados y las cantidad de bytes que ocupan.

***Ejemplo 1***

```c
#include <stdio.h>
void main()
{
	int arraynota[ 9 ],i ;
	char Array [ ]={ 'a','b','c','d','e','f','g','h','i' }
	for (i=0;i<9;i++)
	{
		printf("Ingrese la nota %d:",i+1);
		scanf("%d",&arraynota[ i ]);
	}
	for (i=0;i<9;i++)
	{
		printf("El elemento%d contiene%c y se encuentra en%x y ocupa %d
			:",i,Array[ i ], &Arrai[ i ],sizeof(Array[ i])); /*Esta función
sizeof()retorna el numero de bytes que ocupa  el elemento Array [ i ]*/
	}
	for (i=0;i<9;i++)
	{
		printf("El elemento%d contiene%d y se encuentra en%x y ocupa %d
		:",i,arraynota[ i ], &arraynota[ i ],sizeof(arraynota[ i]));
	}
	return 0;
}
```

OTRO EJEMPLO CON ARREGLOS UNIDIMENCIONALES (VECTORES)
Este programa carga un arreglo con veinte valores enteros calcula el promedio de los valores y el producto de los mismos

***Ejemplo 2***

```c
#include <stdio.h>
main()
{
	int A[20],i,prod=1;
	float prom=0;
	for (i=0;i<20;i++)
	{
		printf("Ingrese el elemento n %d:",i+1);
		scanf("%d",&A[i]);
	}
	for (i=0;i<20;i++)
	{
		prod = prod * A[i];
		prom = prom + A[i];
	}
	prom=prom/20;
	printf("El producto es: %d\nEl promedio es:%f\n\n",prod,prom);
	return 0;
}
```

**¿ Y LOS ARREGLOS BIDIMENCIONALES O MATRICES ?**

Todo lo dicho para los arreglos unidimencionales es valido para los bidimencionales a continuación vamos a ver las pequeñas diferencias.

En el caso de arreglos de doble subíndice por ej:

**mat [ i ] [ j ]**

Donde el valor que le damos a **i** nos indica el renglón o la fila de la matriz comenzando con la fila cero y el valor que le damos a **j** la columna de esta manera podemos ubicar dentro de la matriz o tabla los datos almacenados

En cada uno de los elementos de la misma.

EJERCICIOS EJEMPLOS UTILIZANDO ARREGLOS BIDIMENCIONALES (MATRICES)

En este programa se cargan dos matrices cuadradas de dos por dos y luego se multiplican

***Ejemplo 3***

```
#include <stdio.h>
#define tam 2
/* Programa para multiplicar matrices*/
int main()
{
int A[tam][tam],B[tam][tam], i,j,k;
int C[tam][tam]={ 0 };        //Se define una matriz de 2x2 y la inicializamos a cero
/* Aquí ingresamos las matrices*/
//Matriz A
for(i=0;i<t;am; i++)
      {
      for(j=0;j<t;am;j++)
            {
                printf("Matriz A, elemento %d,%d: ",i+1,j+1);
                scanf("%d",&A[ i ] [ j ]);
            }
      }
//Matriz B
for(i=0;i<t;am;i++)
      {
            for(j=0;j<t;am;j++)
            {
                printf("Matriz B, elemento %d,%d: ",i+1,j+1);
                scanf("%d",&B[ i ] [ j ]);
            }
      }
/* Ahora las multiplicamos*/
      for(i=0;i<tam;i++)
      {
            for(j=0;j<tam;j++)
            {
                for(k=0;k<t;k++)
                        C[ i ] [ j ]=C[ i ] [ j ]+A[ i ] [ k ]*B[ k ] [ j ];
```

```
            }
        }
/*Y mostramos el resultado*/
for(i=0;i<tam;i++)
        {
            for(j=0;j<tam;j++)
            {
                printf("\nMatriz C, elemento %d,%d: %d",i+1,j+1,C[ i ] [ j ]);
            }
        }
        printf("\n\n");
        return 0;
}
```

## Trabajo Practico numero 7

**Actividad 1:**

Diseñar un programa que permita responder a las preguntas ¿Cual es su nombre? y ¿Que edad tiene?. Mostrar el resultado en pantalla.( para almacenar el nombre debe utilizar un arreglo de tipo char por ej : char arraynombre[13] y usar scanf("%s", arraynombre ) no olvidar imprimir las direcciones de memoria donde se almacenan )

**Actividad 2:**

Diseñar un programa que ingrese dos matrices cuadradas al igual que el ejemplo 3 y muestre un menú donde se puedan seleccionar estas distintas opciones, 1-sumar matrices, 2-multiplicar matrices, 3-salir del programa

Y ejecutar las operaciones de acuerdo a la opción elegida. Diagrama de flujo y pseudocodigo

**Actividad 3:**

Codificar en C el ejercicio anterior.

**Actividad 4:**

Diseñar un enunciado para el programa que se encuentra a continuación:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int B[4][5],A[5][4],i=0,j=0;
        for (i=0;i<5;i++)
            for(j=0;j<4;j++)
            {
                printf("\nIngrese %d,%d:",i+1,j+1);
                scanf("%d",&A[i][j]);
            }
        for (i=0;i<5;i++)
        {
            for(j=0;j<4;j++)
            {
            B[j][i]=A[i][j];
```

```
            }
        }
        for (i=0;i<4;i++)
            for(j=0;j<5;j++)
         printf("\nElem. %d,%d: %d",i+1, j+1, B[i][j]);
      system("PAUSE");
      return 0;
}
```

# Ordenamiento y Búsqueda

Dos operaciones muy importantes en programación son el ordenamiento y la búsqueda y son muy a menudo empleadas en programas que procesan gran cantidad de datos. Estos datos se almacenan en los arreglos (unidimencionales o vectores, bidimencionales o Matrices)en forma temporal en la memoria para ser procesados La importancia de estos algoritmos y su análisis reside en que luego podrán ser aplicados a otros tipos de estructuras dinámicas que verán mas adelante en Informática II

## Búsqueda

Esta operación resuelve un problema muy usual que es la búsqueda en un conjunto de datos de un elemento especifico y la recuperación de información asociada al mismo. Por ejemplo si tenemos cargado en un arreglo los números de legajo de los alumnos de primer año de esta facultad, en forma paralela en otra matriz las notas correspondiente a las asignaturas, y en otra los nombres de los alumnos. Al recuperar la ubicación de un legajo en particular, podría con esta ubicación, recuperar la información del nombre y las notas correspondientes a cada una de las asignaturas

| **Filas** | Matriz de caracteres ”nombres” | **Filas** | Matriz de enteros “notas” | | | Nombre de los elementos | Vector legajo |
|---|---|---|---|---|---|---|---|
| Mat[0][j] | Juan Pérez | Not[o][j] | 2 | 8 | 5 | Leg[0] | 1356 |
| Mat[1][j] | Pepe López | Not[1][j] | 9 | 6 | 8 | Leg[1] | 1549 |
| Mat[2][j] | Luis Moya | Not[2][j] | 4 | 8 | 6 | Leg[2] | 2345 |
| Mat[3][j] | Ivi Trento | Not[3][j] | 8 | 7 | 9 | Leg[3] | 7659 |
| Mat[4][j] | Lucas Murua | Not[4][j] | 5 | 9 | 5 | Leg[4] | 6789 |
| Mat[5][j] | Pablo Cayuela | Not[5][j] | 6 | 8 | 6 | Leg[5] | 2367 |
| Mat[6][j] | Melisa Ponce | Not[6][j] | 8 | 7 | 7 | Leg[6] | 8956 |
| Mat[7][j] | Tomas Sosa | Not[7][j] | 9 | 6 | 8 | Leg[7] | 1346 |
| Mat[8][j] | Federico Roca | Not[8][j] | 4 | 5 | 9 | Leg[8] | 1425 |

## Búsqueda Lineal o Secuencial

La búsqueda lineal o secuencial, es la técnica mas simple para buscar un elemento en un arreglo. Este método consiste, en recorrer todo el vector desde el primer elemento hasta el ultimo, de uno en uno o hasta encontrar el elemento buscado. Dicho de otra manera consiste en encontrar la posición de un *elemento clave* en un vector por sucesivas comparaciones de *la clave* con c/u de los datos que contiene el vector.

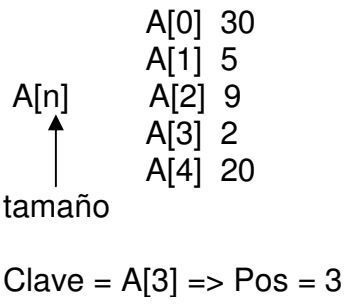
**Ejemplo**
Supongamos una lista de números de legajo incluidos en un array llamado Leg y se desea saber si existe el 2002 y si existe donde esta ubicado

El elemento clave Es "2002" Lo comparamos con cada elemento del vector Leg → hasta ubicar la posición en este caso 5

| | |
|---|---|
| 1254 | Leg[0] |
| 4567 | Leg[1] |
| 4567 | Leg[2] |
| 9876 | Leg[3] |
| 2387 | Leg[4] |
| 2002 | Leg[5] |
| 2078 | Leg[6] |
| 3456 | Leg[7] |
| 1395 | Leg[8] |

**Algoritmo**

```
Encontrado = Falso
Posición = 0
i = 0
Mientras ( i < h ) && ( Encontrado = Falso )
   { Si ( A[ i ] = Clave )
       {
           Posición = i
           Encontrado = Verdadero
       }
     i = i + 1
  }
```




Clave = A[3] => Pos = 3

## Trabajo Practico numero 8

**Actividad 1:**

Completar el programa codificado que se encuentra a continuación .El programa deberá ingresar un arreglo de 10 elementos una variable llamada clave donde se ingresará el valor buscado ,si el valor es encontrado deberá mostrar la posición si no se encuentra el programa deberá mostrar el mensaje correspondiente.

```
#define tam 5
#include<stdio.h>

main( )
{
    int Encont = 0, A[ tam ], i , clave;
    int Pos = 0;
    ____________________________

        Ingreso del vector
```

```
        Ingreso de la clave
    ____________________________

    while ( i < tam && Encont = 0 )
    {
        if (A [ i ] == clave )
        {
            Pos = i;
            Encont = 1;
        }
        i ++;
    }
      ____________________________

        Mostrar clave y posición
      ____________________________
}
```

**Actividad 2:**
Diseñar un programa que ingrese el numero de legajo(en un arreglo) y la nota del primer parcial de informática (en otro arreglo) de cada alumno del curso 1R3 .El programa deberá ser capaz de mostrar según una cierta nota ingresada quienes son los que obtuvieron esa nota para ello tendrá que almacenar en un arreglo el numero de legajo de los alumnos que tengan esa nota del primer parcial y luego mostrarlo.
Diagrama de flujo y pseudocódigo

**Actividad 3:**
Codificar en C el ejercicio anterior

# Búsqueda Binaria

La finalidad de este método es la misma que para la búsqueda secuencial pero el método es totalmente diferente. Se parte de un vector previamente ordenado, luego se determina el elemento central que divide en dos partes al vector. Los elementos o datos que se encuentran por encima del elemento central, son mayores que él, por ej. El A [ 6 ] es mayor que el A [ 5 ] y también sucede lo inverso, el A [ 4 ] es menor que el A [ 5 ], por lo tanto si comparamos la clave con el elemento central, podremos determinar:

- Si es = entonces habremos encontrado la posición.
- Si es < entonces el dato clave se encuentra en la primera mitad del vector, o sea entre 0 y central – 1
- Si es > entonces el dato clave se encuentra en la segunda mitad del vector, o sea entre central + 1 y última pos.

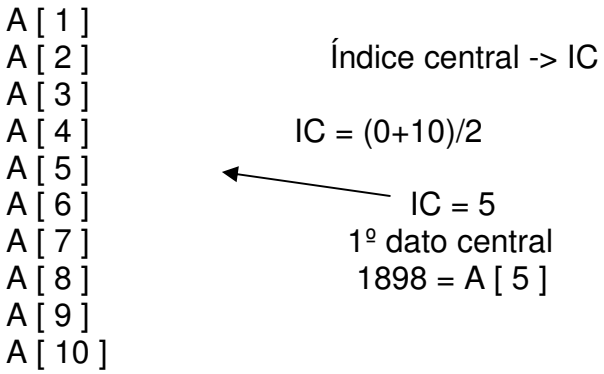
Luego si el elemento no es encontrado, se recalcula el elemento central de la parte en cuestión y así a través de estoa tanteos, se determina la posición del elemento clave sin necesidad de comparar la clave 1 a 1 con los elementos del vector produciendo de ésta manera una búsqueda más eficiente.

**Ejemplo Lista ordenada**

Supongamos que clave = 1555,
Comparamos dato central
con clave

A [ 5 ] Clave

| | |
|---|---|
| A [ 0 ] | 1331 |
| | 1337 |
| | 1555 |
| | 1750 |
| | 1892 |
| | 1898 |




**Algoritmo**

```
Primero = 0
Ultimo = tam – 1
Encontrado = falso
Mientras ( Primero < Ultimo ) && ( Encontrado == falso )
   {
   Central = ( Primero + Ultimo ) / 2
   Si ( Clave == A [ Central ] )
      Encontrado = verdadero
   Sino
      Si ( Clave > A [ central ] )
         Primero = Central + 1
      Sino
         Ultimo = Central - 1
   }
Si ( Encontrado == verdadero )
   Primero = Central
Sino
   Imp “No se ha encontrado”
```

## Trabajo Práctico numero 9

**Actividad 1:**

Diseñar un programa que ingrese un arreglo de 20 elementos ordenados y una variable llamada clave donde se ingresará el valor buscado ,si el valor es encontrado el programa deberá mostrar la posición ,si no se encuentra, deberá dar el mensaje correspondiente. Aplicar el algoritmo de búsqueda Binaria. Diagrama de flujo y pseudocódigo.

**Actividad 2:**

Codificar en C el ejercicio anterior.

# Ordenamiento

Existen varios y complicados métodos de ordenamiento (que tienen una mejor eficiencia) así como de búsqueda Debido a que este es un curso inicial vamos a ver dos, el método de la burbuja y la burbuja mejorado que son los más simples

# Método de la burbuja

El nombre de este método proviene del hecho físico que la burbujas de aire al ser un elemento mas liviano suben y que los elementos mas pesados caen. El método consiste en realizar una comparación sucesiva de los elementos consecutivos el vect[ j ] con el vect[ j+1] si el primero es mayor que el segundo se produce el intercambio de posiciones de esta manera el elemento mas pesado cae a una posición superior y el mas liviano sube a una posición inferior de esta manera al terminar la pasada el ultimo elemento quedo ordenado así vemos que por cada pasada queda ordenado un elemento mas dentro del arreglo por lo tanto para ordenarlo completamente vamos a tener que ejecutar, siendo un arreglo de N elementos, N-1 pasadas

$\begin{bmatrix}23\\12\\21\\9\end{bmatrix}$ $\begin{bmatrix}12\\23\\21\\9\end{bmatrix}$ $\begin{bmatrix}12\\21\\23\\9\end{bmatrix}$ $\begin{bmatrix}12\\21\\9\\23\end{bmatrix}$ Pasada 0

$\begin{bmatrix}12\\21\\9\\23\end{bmatrix}$ $\begin{bmatrix}12\\21\\9\\23\end{bmatrix}$ $\begin{bmatrix}12\\9\\21\\23\end{bmatrix}$ $\begin{bmatrix}12\\9\\21\\23\end{bmatrix}$ Pasada 1

$\begin{bmatrix}12\\9\\21\\23\end{bmatrix}$ $\begin{bmatrix}9\\12\\21\\23\end{bmatrix}$ $\begin{bmatrix}12\\9\\21\\23\end{bmatrix}$ $\begin{bmatrix}12\\9\\21\\23\end{bmatrix}$ Pasada 2

```
for ( i = 0; i < tam – 1; i ++)
    {
    for ( j = 0; j < ( tam – 1 ); j ++ )
        {
        if ( vect [ j ] > vect [ j + 1] )
          {
          aux = vect [ j ];
          vect [ i ] = vect [ j + 1];
          vect [ j + 1] = aux;
          }
        }
    }
```

# Método de la burbuja mejorado

$\begin{bmatrix}23\\12\\21\\9\end{bmatrix}$ $\begin{bmatrix}12\\23\\21\\9\end{bmatrix}$ $\begin{bmatrix}12\\21\\23\\9\end{bmatrix}$ $\begin{bmatrix}12\\21\\9\\23\end{bmatrix}$ Pasada 0

$\begin{bmatrix}12\\21\\9\\23\end{bmatrix}$ $\begin{bmatrix}12\\21\\9\\23\end{bmatrix}$ $\begin{bmatrix}12\\9\\21\\23\end{bmatrix}$ Pasada 1

```
for ( i = 0; i < tam – 1; i ++)
    {
    for ( j = 0; j < ( tam – 1 – i ); j ++ )
        {
        if ( vect [ j ] > vect [ j + 1] )
          {
          aux = vect [ j ];
          vect [ j ] = vect [ j + 1];
          vect [ j + 1] = aux;
          }
        }
```

$\begin{bmatrix}12\\9\\21\\23\end{bmatrix}$ $\begin{bmatrix}9\\12\\21\\23\end{bmatrix}$ Pasada 2

## Trabajo Práctico numero 10

**Actividad 1:**

Analizar cuales son las diferencias entre los dos métodos planteados en forma de algoritmo y describir el ultimo método verbalmente.

**Actividad 2:**

Diseñar un programa que ingrese un arreglo de 20 elementos y lo ordene Utilizando el método de la burbuja mejorado el programa deberá mostrar el arreglo inicial y a continuación el arreglo ordenado. Diagrama de flujo y pseudocódigo

**Actividad 3:**

Codificar en C el programa anterior.