Arrays.

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Pablo Gil*
Hector Farina
Ing. Ernesto Gigliotti

Revisores: *Ing. Ernesto Gigliotti*
Lic. Mauricio Dávila

Versión : 2.1

Índice de contenido

1 Arrays

Un vector o array es un conjunto de elementos del mismo tipo (enteros, flotantes, caracteres) que se agrupan bajo un mismo nombre y se diferencian entre sí por un índice.
La forma genérica de declararlo en un programa es la siguiente:

```
Tipo  nombre[cantidad]
```

Donde tipo es el tipo de dato, por ejemplo *int, float, char*, etc.
"nombre" es el nombre del vector y "cantidad" es la cantidad de elementos que contiene el vector.
Un ejemplo de esto puede ser

```
int   v[5];          defino un vector que se llama v de 5 enteros
float  vec[10];      defino un vector que se llama vec de 10 flotantes
char   v1[5];        defino un vector llamado v1 de 20 caracteres
```

La representación gráfica del vector "v" puede ser la siguiente:

Valor:	12	23	9	37	31
Índice:	**0**	**1**	**2**	**3**	**4**

Como se puede ver el índice comienza siempre en cero y va hasta la cantidad de elementos menos uno.
Para acceder a un elemento cualquiera de vector, por ejemplo al tercer elemento se escribe:

```
v[2]
```

Donde "v" es el nombre del vector y entre corchetes se le pone el índice.

De acuerdo a esto, decimos que el vector tiene 5 enteros, el nombre del vector es "v" y cada uno de sus elementos serán

- v[0] tiene el valor 12 y es el primer elemento del vector
- v[1] tiene el valor 23 y es el segundo elemento del vector
- v[2] tiene el valor 9 y es el tercer elemento del vector
- v[3] tiene el valor 37 y es el cuarto elemento del vector
- v[4] tiene el valor 31 y es el quinto elemento del vector

Se observa que entre el número de elemento y el índice existe una diferencia de 1
Note que v[5]no existe como elemento del vector.

Para llenar un vector con valores (cargar), existen 2 formas que son la secuencial y la aleatoria.

En la carga secuencial se carga el vector ordenadamente, es decir el primero después el segundo, el tercero y así hasta llegar al último.

En la carga aleatoria no existe un orden establecido, por lo tanto se debe informar el número a cargar y en que posición del vector se carga.

1.1 Carga secuencial

Como se comentó anteriormente, la carga secuencial es ordenada y de acuerdo a esto deberemos llenar el vector respetando el siguiente orden:

v[0], v[1], v[2], v[3], v[4]

Como se observa, el que cambia en forma creciente es el índice del vector que es el que nos permite hacer la carga en forma secuencial.
Para llevar esto a código de programa, debemos pensar en una estructura de control que nos permita realizar un número prefijado de repeticiones y que me incremente una variable en una unidad. Dicha estructura de control puede ser el bucle *for*.

Un ejemplo de programa que nos permite cargar un vector en forma secuencial es el siguiente:

```
#include <stdio.h>

void main(void)
{
	int v[5],i;

	for (i=0;i<5;i++)
	{
		printf("Ingrese valor a cargar en el vector");
		scanf("%d",&v[i]);
	}
}
```

En el programa la variable "i" del bucle *for* va a tomar valores desde 0 a 4 ordenadamente. Cuando comienza el *for* la variable "i" vale 0 (i=0), se cumple con la condición de que i<5 y entonces entramos al *for*. Se ejecuta el *printf* y luego el *scanf* lee desde el teclado y carga el valor en v[i] (v[0]). Se vuelve al *for* incrementando i en 1 (i++) por lo tanto el valor de "i" ahora es 1 y se cargará v[1], siguiendo i=2 y v[2], i=3 y v[3] para finalmente i=4 y v[4].
De esta forma se cargan todos los elemento del vector en forma secuencial.

Ejemplo: De los 100 empleados de una fábrica se registra número de legajo (coincidente con el índice), edad y salario. Se pide ingresar los datos consecutivamente y calcular el sueldo promedio.

Comenzamos el problema analizando donde se guardarán los datos.
Sabemos que se ingresa legajo, edad y sueldo de 100 empleados y que el legajo coincide con el índice, por lo tanto debemos definir 2 vectores paralelos, uno será de enteros (para la edad) y el otro será de flotantes (para el sueldo). El legajo va de uno a 100 y nosotros deberemos establecer la relación que hay con el índice, que sabemos que va de cero a 99 en el vector. Aquí se nota la diferencia de uno entre el legajo e índice.

Un esquema de los vectores puede ser el siguiente:

Edad					

Sueldo					

La relación entre el índice y el legajo será la siguiente:

Índice	0	1	2	3	4
Legajo	1	2	3	4	5

Sabiendo que se cargan los datos en forma consecutiva, sabemos que la carga deberá ser secuencial por legajo, por lo tanto usamos la estructura del *for* que se escribió arriba

```
for (i=0;i<100;i++)
{
	printf("Ingrese edad para el legajo %d",i+1);
	scanf("%d",&edad[i]);
	printf("Ingrese sueldo para el legajo %d",i+1);
	scanf("%f",&sueldo[i]);

}
```

Como se ve en el código, el legajo no se ingresa por que lleva relación directa con el índice que en este caso es la variable "i".
Una vez cargados los 100 datos debemos calcular el sueldo promedio. Para ello se deben sumar todos los sueldos que están en el vector y luego dividirlos por la cantidad de elementos que en este caso es 100.

Para realizar la suma utilizamos un *for* ya que conocemos la cantidad de iteraciones.
Finalmente se muestra en pantalla el sueldo promedio.

El programa completo será el siguiente:

```
#include <stdio.h>
#define MAX 100

void main(void)
{
	int edad[MAX],i;
	float sueldo[MAX],suma=0,prom,aux;

	for(i=0;i<MAX;i++)
	{
		printf("Ingrese la edad del legajo %d: ",i+1);
		scanf("%d",&edad[i]);
		printf("Ingrese el sueldo del legajo %d: ",i+1);
		scanf("%f",&aux);
		sueldo[i]=aux;
	}

	for(i=0;i<MAX;i++)
		suma=suma+sueldo[i];

	prom=suma/MAX;
	printf("El sueldo promedio es %.2f",prom);
}
```

En la carga cuando i=0 se esta pidiendo el ingreso de la edad del legajo 1, pero se guarda en edad[0]. Lo mismo ocurre con el sueldo y con el resto de los ingresos.

I	Edad ingresada	Sueldo ingresado		Vector Edad		Vector Sueldo	Legajo
0	21	500		21		500.00	1
1	18	470.20		18		470.20	2
2	30	890.75		30		890.75	3
3	29	756.20		29		756.20	4

Se observa que el promedio se calcula recién cuando termina el *for*, es decir cuando se tiene la suma completa. Si se pone dentro del for, en lugar de calcular el promedio 1 sola vez se calcularía 100 veces.

1.2 Carga aleatoria

De acuerdo a lo comentado con anterioridad para la carga aleatoria se debe ingresar 2 valores que son la posición donde queremos guardar el número y luego el número que queremos cargar.
Puede ocurrir que en algún caso no se carguen todos los elementos del vector o que se pretenda cargar algún valor en un lugar ya cargado. Para analizar esto comenzaremos por el caso más simple y dejando que el usuario decida cuando terminar de cargar elementos en el vector. De acuerdo a esto, como no se sabe cuántos elementos se van a cargar ni en qué orden, debemos primero inicializar todos los elementos del vector en un valor conocido que normalmente es cero.

Seguidamente se debe usar un ciclo de control en el cuál la cantidad de iteraciones que se realicen quede fijada por alguna condición que acepte el usuario, en este caso el ciclo que más se adecua es el *while* o *do while*.

Un ejemplo de programa puede ser el siguiente:

```
#include <stdio.h>

void main(void)
{
	int v[5],pos,i;
	char seguir;

	for(i=0;i<5;i++)
		v[i]=0; // inicializamos vector
	do
	{
		printf("Ingrese posici¢n");
		scanf("%d",&pos);
		printf("Ingrese valor a cargar en el vector");
		scanf("%d",&v[pos]);
		printf("Desea ingresar otro dato S/N?");
		scanf("%c",&seguir);
	}while(seguir=='s');
}
```

Una vez inicializado el vector se ingresa al ciclo *do while* y supongamos que se ingresa la posición 2 y como valor a cargar 15, con lo cual tendremos:

pos=2 y v[pos]=15 v[pos] será v[2] y el vector queda:

0	0	15	0	0

Contestamos que si a ingresar otro dato y ahora pos=4 y el valor a cargar 43, quedando el vector:

0	0	15	0	43

Cabe destacar cómo se cargan los datos de acuerdo a la posición que el usuario elige y remarcar que no es en forma ordenada. Solo será ordenada si se ingresan las posiciones en forma consecutiva.
Cuando se conteste que no se quieren ingresar mas datos, se finaliza la carga.

Ejemplo: De los 100 empleados de una fábrica se registra número de legajo (coincidente con el índice), edad y salario. Se pide ingresar los datos y calcular el sueldo promedio.
El análisis del problema se plantea exactamente igual al caso de carga secuencial. La única diferencia es que ahora se debe ingresar el legajo quien va a ser el que me indique la posición en donde se va a cargar el sueldo y la edad.

```
#include <stdio.h>
#define MAX 100
void main(void)
{
	int edad[MAX],i,leg;
	float sueldo[MAX],suma=0,prom,aux;
	char seguir;

	for(i=0;i<MAX;i++)
	{
		edad[i]=0;
		sueldo[i]=0;
	}
	do
	{
		printf("Ingrese Legajo");
		scanf("%d",&leg);
		printf("Ingrese Edad");
		scanf("%d",&edad[leg-1]);
		printf("Ingrese el sueldo");
		scanf("%f",&aux);
		sueldo[i]=aux;

		printf("Desea ingresar otro dato S/N?");
		scanf("%c",&seguir);
	}while(seguir=='s');

	for(i=0;i<MAX;i++)
		suma=suma+sueldo[i];

	prom=suma/MAX;
	printf("El sueldo promedio es %.2f",prom);
}
```

Se debe recordar que la relación entre el índice y el legajo es legajo=indice + 1.
El primer legajo que se cargó fue el 3 que corresponde al índice 2, por lo tanto los primeros datos que se cargaron fueron el 21 y el 500, luego el 18 y 470.20 y asi sucesivamente. La flecha al costado izquierdo de la tabla indica el órden en el que se cargaron los datos.

Legajo ingresado	Edad ingresada	Sueldo ingresado		Vector Edad		Vector Sueldo	Indice
3	21	500		30		890.75	0
4	18	470.20		29		756.20	1
1	30	890.75		21		500.00	2
2	29	756.20		18		470.20	3

Observamos que el legajo se carga pero no se guarda en el vector, sirve para posicionarse dentro de uno de los elementos del vector.

El resto del programa sigue igual que en el caso de carga secuencial. Solo es secuencial o aleatoria la carga, el resto de las operaciones con vectores normalmente se hace en forma secuencial usando un *for*.

1.3 Búsquedas y ordenamiento sobre vectores

Vamos a desarrollar un programa de ejemplo con distintas tareas que se necesitan realizar sobre vectores.

```
#include <stdio.h>

#define MAX  5

void main(void)
{
	int vec[MAX],i,mayor,menor,num,flag;

	//Carga el vector

	for(i=0;i<MAX;i++)
	{
		printf("Ingrese nro");
		scanf("%d",&vec[i]);
	}
```

En este bucle “for”, vamos pidiendo al usuario que ingrese números y los guardamos en las posiciones del array.

1.3.1 Búsqueda del mayor

```
//buscar mayor

mayor=vec[0];
for(i=0;i<MAX;i++)
{
	if(vec[i]>mayor)
	{
		mayor=vec[i];
		/* otras asignaciones */
	}
}
```

En "otras asignaciones" vienen todas aquellas asignaciones relacionadas con la busqueda, por ejemplo buscar el legajo de la persona de mayor edad. Se busca la edad mayor y dentro del bloque se agrega el legajo. Mayor_leg=legajo[i]

Para encontrar el mayor elemento dentro de un vector lo primero que hacemos es tomar uno cualquiera de los elementos del mismo y suponer que es el mayor. Luego comparando cada elemento del vector contra el valor de la variable ***mayor***, que es actualizado mientras se recorre el vector , se llega al final del vector con el mayor elemento guardado en la variable ***mayor***

Considere un vector con los siguientes datos cargados donde se pretende encontrar el elemento mayor y la posición en la cual esta ubicado.

5	**1**	**9**	**8**
[0]	**[1]**	**[2]**	**[3]**

en la primer línea mayor=vec[0]; estamos asignando a la variable mayor el primer elemento del vector, es decir que estamos inicializando la variable mayor con un valor de 5 seguidamente seteamos la posicion inicial para dicha variable.

El ciclo *for* es el que va a permitir recorrer todo el vector realizando MAX iteraciones. La decisión se hace con el *if* en donde tendremos

```
I=0    mayor=5          5>5  Falso
I=1    mayor=5          1>5  Falso
I=2    mayor=5          9>5  Verdadero  -> mayor=9  -> posicion=2;
I=3    mayor=9          8>9  Falso
```

Terminado el *for* nos queda la variable mayor con el valor 9, que es el mayor de los números dentro del vector.

1.3.2 Búsqueda del menor

```
//buscar menor

menor=vec[0];
for(i=0;i<MAX;i++)
{
	if(vec[i]<menor)
	{
		menor=vec[i];
		/* otras asignaciones */
	}
}
```

En "otras asiganciones" vienen todas aquellas asignaciones relacionadas con la búsqueda, por ejemplo buscar el legajo de la persona de menor edad. Se busca la edad menor y dentro del bloque se agrega el legajo. Menor_leg=legajo[i]

El razonamiento es el mismo que para la búsqueda del mayor. Solo cambia el signo de la desigualdad: **`if (vec[i]< menor)`**

1.3.3 Búsqueda de un número dentro del vector

```
//buscar un numero dentro del vector

flag=0;
printf("Ingrese numero a buscar");
scanf("%d",&num);
for(i=0;i<MAX;i++)
{
	if(num==vec[i])
	{
		/* asignaciones y modificaciones */
		printf("Se encontro el numero");
		flag=1;
	}
}
if(flag==0)
	printf("El numero no se encontro");
```

Para buscar un número dentro de un vector, lo debemos recorrer y preguntar si el elemento del vector es igual al número ingresado.

De acuerdo a esto surge que para recorrer el vector debemos colocar un *for* y dentro del mismo un if para preguntar por la igualdad.

Las 2 primeras líneas de código solicitan el ingreso del número a buscar. Analicemos el funcionamiento con un ejemplo del vector cargado:

5	1	9	8
[0]	[1]	[2]	[3]

Ahora supongamos que el número que se quiere buscar es el 9, con lo cual la variable "num" tendrá el valor 9. Si seguimos el código tendremos

```
I=0    vec[i]=5      if(num==vec[i])  Falso
I=1    vec[i]=1      if(num==vec[i])  Falso
I=2    vec[i]=9      if(num==vec[i])  Verdadero
```

Al entrar al *if* se muestra el mensaje y el *break* hace que se termine el *for* .
Se observa que cuando el número se encuentra en el vector se termina la búsqueda.
Nos queda analizar una pequeña modificación al algoritmo para que me permita informar si el número no existe en el vector.

El problema a resolver es cómo se da cuenta el programa que el número no está en el vector. La respuesta es EL PROGRAMA NO TIENE FORMA DE SABER QUE EL NÚMERO NO ESTA EN EL VECTOR, pero si sabe cuando lo encuentra.

De acuerdo a esto podemos decir que si el programa nunca entro al *if* (despues de recorrer todo el vector) significa que no lo encontró y entonces la solución es colocar una bandera dentro del *if*. El código es el siguiente:

```
flag=0; // Supongo que el número no lo encuentro

printf("Ingrese numero a buscar");
scanf("%d",&num);

for(i=0;i<MAX;i++)
{
	if(num==vec[i])
	{
		printf("Se encontro el numero");
		flag=1; // si lo encontro, pongo el flag en 1
	}
}

if(flag==0)
	printf("El numero no se encontro");
```

1.3.4 Ordenamiento del vector

Método de burbujeo

Una de las formas más usadas para ordenar un vector se llama Método de burbujeo.

Se van comparando los elementos del vector, que en forma genérica será el elemento "i" (vec[i]) con el resto de los elementos del vector (desde la posición siguiente a "i" hasta el final del mismo).

Para ello, se utiliza un primer bucle "for" que permite iterar el vector y obtener sus elementos, desde el comienzo hasta el final (utilizando "i") para cada elemento obtenido, se itera nuevamente el vector desde la posición siguiente al elemento actual, hasta el final, para ello se utiliza un segundo bucle "for" con una variable "j" que comienza valiendo i+1.

```
for(i=0;i<FIL-1;i++)
{
	for(j=i+1;j<FIL;j++)
	{
		if(vec[i]>vec[j])
		{
			aux=vec[i];
			vec[i]=vec[j];
			vec[j]=aux;
		}
	}
}
```

En la tabla que sigue se hace un esquema de cómo va evolucionando el vector a medida que avanzan los 2 bucles *for*

i	j	vec[i]	vec[j]	if(vec[i]>vec[j])	vec
0	1	10	50	FALSO	10 60 50 30
0	2	10	60	FALSO	10 60 50 30
0	3	10	90	FALSO	10 60 50 30
1	2	60	50	VERDADERO	10 50 60 30
1	3	50	30	VERDADERO	10 30 60 50
2	3	60	50	VERDADERO	10 30 50 60

Cuando se da la condición verdadera del *if* se intercambian los elementos, para lo cuál se necesita un auxiliar. Si el auxiliar no se utiliza uno de los 2 valores se perdería.

Método de burbujeo más eficiente

Como se observa en el código anterior, para cualquier condición, el segundo bucle con todas sus iteraciones siempre se ejecuta una cierta cantidad de veces, que es igual al tamaño del array-1.

Si el vector ya se encuentra ordenado, igualmente el segundo bucle junto con todas sus iteraciones se ejecuta FIL-1 veces (siendo FIL el tamaño del array). Esto quiere decir que el algoritmo anterior posee muchas iteraciones redundantes, y nunca "se da cuenta" que ya no hay nada que ordenar.

Si el array ya estuviera ordenado, nunca se ejecutaría el código dentro de la sentencia *if* que realiza la comparación, ya que la comparación siempre sería falsa. Bajo esta premisa, podemos modificar el algoritmo, haciendo que el segundo bucle *for* se ejecute **la cantidad de veces que sean necesarias** hasta que el array este ordenado, y no más. Para ello, cambiamos el primer bucle *for* por un bucle *while*, y utilizamos un *flag* para saber si ya terminamos de ordenar.

```
void bubbleSort (int a[],int len)
{
    int j, aux;
    int flagNoEstaOrdenado = 1;
    while (flagNoEstaOrdenado==1)
    {
         flagNoEstaOrdenado = 0;
        for (j = 1; j < len; j++)
        {
            if (a[j] < a[j - 1])
            {
                aux = a[j];
                a[j] = a[j - 1];
                a[j - 1] = aux;
                flagNoEstaOrdenado = 1;
            }
        }
    }
}
```

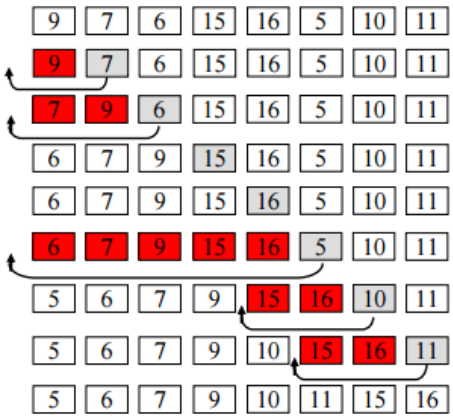
Como se observa en el ejemplo, utilizamos un bucle *for* para iterar todos los elementos y compararlos entre sí. ¿Cuántas veces se ejecutará este bucle *for* completo con todas sus iteraciones?: la cantidad de veces que sea necesaria hasta que el array quede ordenado. ¿Cómo sabemos que el array está ordenado? Porque nunca entró al código dentro del *if*, esto quiere decir que se recorrió todo el array una vez y no se encontraron ítems desordenados, bajo este condición, el flag permanece en "0" y cuando se vuelve a ejecutar la evaluación del flag en el bucle *while*, el mismo terminará.

Método de inserción

A pesar de la mejora, el método anterior es uno de los más ineficientes que existen, y su uso no es recomendado.Un algoritmo más eficiente de ordenamiento es el ordenamiento por inserción.

La idea del algortimo, es ir tomando cada uno de los items del array (empezando desde el segundo) y comparándolo con todos los items que están a su izquierda. Mientras el ítem sera menor el que ítem a su izquierda se seguirá comparando. **En el ítem tomado, será "movido o insertado" de modo que a su izquierda no tenga ítems mayores.** En el caso de no encontrar uno mayor, nuestro ítem no se moverá.

Ejemplo:



El algoritmo comienza tomando el segundo ítem (el 7) y comienza a compararlo con todos los ítems a la izquierda del mismo (en este caso el 9) como es menor que el 9, y no hay más ítems a la izquierda, el 7 se inserta a la izquierda del 9 (y al comienzo del array)

Como el segundo ítem ya está ordenado, se toma el siguiente (el tercero) es decir el 6. Se comienza a comparar con los ítems a la izquierda. Como el 6 es menor que 9 y es menor que 7, el 6 se "inserta" a la izquierda del 7 (y al comienzo del array)

Ahora se toma el cuarto ítem (que sería el 15) y se compara con los ítems a su izquierda, como no es menor que el 9 o que el 7 o que el 6, no se mueve.

Luego se toma el quinto ítem (que sería el 16) y se compara con los ítems a su izquierda, como no es menor que el 15 o que el 9 o que el 7 o que el 6, no se mueve.

A continuación se toma el sexto elemento (el 5) y se compara con los ítems a su izquierda, como es menor que todos, se inserta al comienzo.

Al tomar luego el ítem septimo (que sería el 10) y se comienza a comparar, el mismo es menor que el 16 y que el 15, pero se encuentra que él es es mayor que el 9, por lo que el 10 debe insertarse a la izquierda del último mayor encontrado (del 15).

Por último al tomar el 11, se comienza a comparar y se encuentra que a su izquierda tiene el 16 y luego el 15 como mayores, pero luego encuentra un menor, el 10, por lo que el 11 se inserta a la izquierda de su último mayor (el 15).

El código del algotimo puede observarse en la siguiente función:

```
void insertion(int data[],int len)
{
    int i,j;
    int temp;
    for(i=1;i<len;i++)
    {
       temp = data[i];
       j=i-1;
       while(j>=0 && temp<data[j]) // temp<data[j] o temp>data[j]
       {
          data[j+1] = data[j];
          j--;
       }
       data[j+1]=temp; // inserción
    }
}
```

El primer bucle itera todos los ítems del array, comenzando del segundo (i=1) hasta el final del mismo. Por cada ítem, se evalúan los ítems a su izquierda (utilizando j que comienza en i-1 y se va decrementando hasta 0, o hasta que se encuentre un ítem menor).

El ítem que evaluamos e insertamos, lo guardamos en "temp" y lo vamos comparando con los valores a su izquierda, utilizando "data[j]". Mientras el ítem sea menor que el ítem a si izquierda, desplazamos todos los ítems a la derecha, para dejar lugar para que "temp" sea insertado en la posición que corresponda. La inserción ocurre al finalizar el bucle *while*.