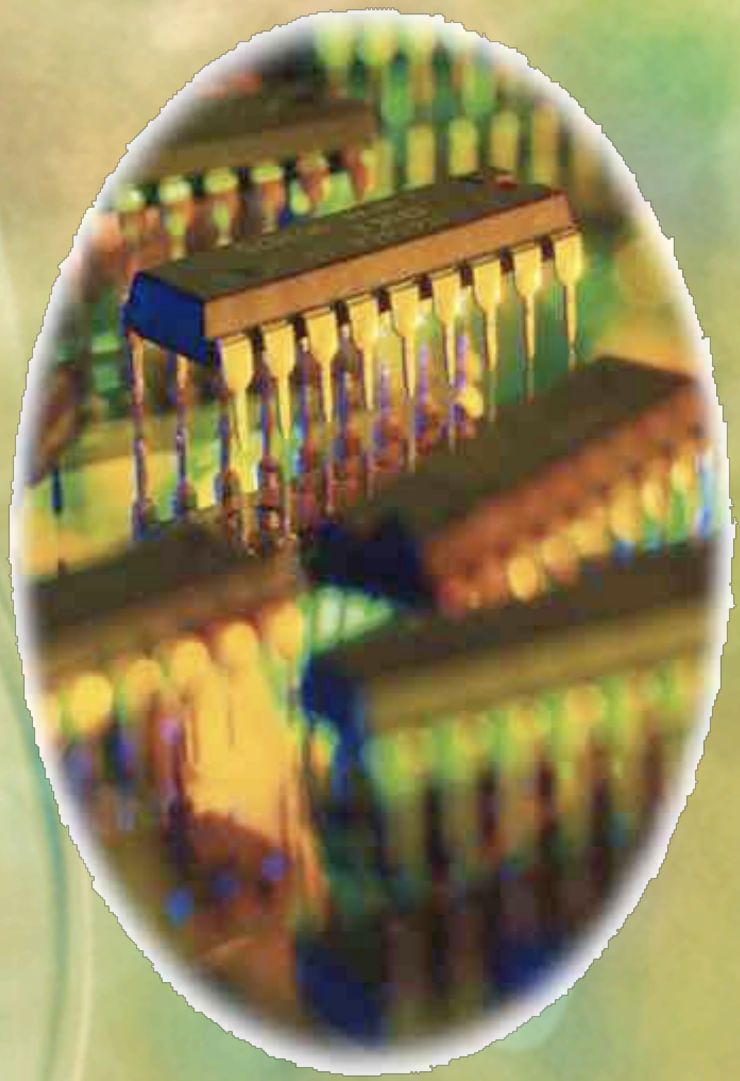
TEMA

5

Vectores y Matrices



Dept. Ciencias de la Computación e I.A.

Universidad de Granada

VECTORES
Y
MATRICES

- Motivación.
- Operaciones Básicas.
- Paso de vectores como parámetros.
- Búsqueda y Ordenación.
- Matrices.

Motivación

- En casi todos los problemas, es necesario mantener una relación entre variables diferentes o almacenar y referenciar variables como un grupo. Para ello, los lenguajes ofrecen tipos más complejos que los vistos hasta ahora.
- Un tipo de dato compuesto es una composición de tipos de datos simples (o incluso compuestos) caracterizado por la organización de sus datos y por las operaciones que se definen sobre él.
- Una matriz es un tipo de dato, compuesto de un número fijo de componentes del mismo tipo y donde cada una de ellas es directamente accesible mediante un índice. Un vector es una matriz de una dimensión.

Un Ejemplo

```
/*Programa para realizar la media de TRES notas, y
  calcular cuántos alumnos han superado la media */
#include <stdio.h>
int main(){
  int cuantos=0;
  double nota1, nota2, nota3, media;
  /*leer nota1, nota2, nota3*/
  . . . . . .
  media = (nota1+nota2+nota3)/3.0;
  if (nota1 > media) cuantos++;
  if (nota2 > media) cuantos++;
  if (nota3 > media) cuantos++;

  printf("Media Aritmetica = %lf \n",media);
  printf("%ld alumnos han superado la media\n",
   cuantos);
  return 0;}
```

¿Qué sucede si queremos almacenar las notas de 150 alumnos?

Número de variables imposible de sostener y recordar

Solución: Introducir un *tipo de dato nuevo* que permita representar dichas variables en una única estructura de datos

	Notas[1]	Notas[2]	Notas[3]
NOTAS	3.8	5.7	6.5

El tipo nuevo será una matriz o vector.

Se declara una variable ***notas*** de tipo vector y cada componente se accede en la forma:

notas[indice]

Declaración

<tipo><identificador>[<CantValores>];

donde

<tipo> indica el tipo de dato común a todas las componentes del vector.

<CantValores> determina el número de componentes del vector. Puede ser un literal entero o una constante entera.

Ejemplos

```
double notas[3];
int casados[40];
char temp[100];
```

Operaciones Básicas

Acceso:

<identificador>[indice]

0 ≤ Indice ≤ CantValores-1

Dada la declaración:

double v[3];

Cada componente del vector es una variable de tipo double.

v[2] , v[0] son variables de tipo double

v['3'] , v[9] no son accesos correctos.

Operaciones Básicas

Asignación de Valores:

<identificador>[indice] = <Expresion>

Donde *<Expresión>* ha de ser del mismo tipo que el definido en la declaración del vector (o al menos compatible).

v[0] = 3.45; *v[1] = sqrt(256);*

No se permiten asignaciones globales sobre todos los elementos del vector, salvo en el momento de la definición de la variable, la inicialización. Las asignaciones se deben realizar componente a componente.

Mecanismos de Inicialización

C permite inicializar una variable de tipo vector, en la declaración.

int v[3]={4,5,6};

inicializa v[0]=4, v[1]=5 y v[2]=6

int v[7]={3,5};

inicializa v[0]=3 v[1]=5 y el resto no se inicializan.

int v[]={1,3,9};

automáticamente el compilador asume *int v[3]*

Recorriendo Vectores

La lectura y/o escritura de los valores de un vector, se realiza componente a componente (salvo en la inicialización).

Para recorrer las componentes, utilizaremos normalmente el siguiente esquema:

```
#define TAM  20
int main(){
 int vec[TAM];
 int i;
 for(i=0; i < TAM; i=i+1){
    manipulo vec[i];
 }
 return 0;
}
```

Búsqueda lineal de un Elemento

```
#define TAM 100
int main(){
 int v[TAM];
 int i, posicion, elemento;
 int Encontrado=0;
 /*supongamos que v ya tiene valores*/
 printf("Ingrese el elemento a buscar:”);
 scanf(“%d”,&elemento);

 for (i=0; i<TAM; i++){
   if (v[i] == elemento) {
       posicion=i; Encontrado=1;
    }
 }
if (Encontrado)
   printf("Encontrado en la posición %d \n”,posicion);
else
   printf("No Encontrado \n”);
return 0;
}
```

Búsqueda lineal de un Elemento 2

```
#define TAM 100
int main(){
 int v[TAM];
 int i, posicion, elemento;
 int Encontrado=0;

 /*supongamos que v ya tiene valores*/
 printf("Ingrese el elemento a buscar:");
 scanf("%d",&elemento);
 i=0;
 while ((i<TAM) && !Encontrado)
    if (v[i] == elemento){
         posicion=i;
        Encontrado=1;
    }
    else
        i++;
 if (Encontrado)
   printf("Encontrado en la posición %d \n",posicion);
 else
   printf("No Encontrado \n");
 return 0;}
```

Búsqueda binaria de un Elemento (Vector ordenado)

```
#define TAM 100
int main(){
 int v[TAM];
 int elemento, posicion, inicio=0, final=TAM-1;
 int Encontrado=0;

 /*supongamos que v ya tiene valores y están ordenados*/
 printf("Ingrese el elemento a buscar:”);
 scanf(“%d”,&elemento);
 while (inicio<=final){
    medio=(inicio+final)/2;
    if (v[medio] == elemento) {
         posicion=medio;
         Encontrado=1;
    }else if (elemento>v[medio])
             inicio=medio+1;
          else
             final=medio-1;
 }
 if (Encontrado) printf("Encontrado en la posición %d \n”,posicion);
 else printf("No Encontrado \n”);
 return 0;}
```

Frecuencia de un Elemento

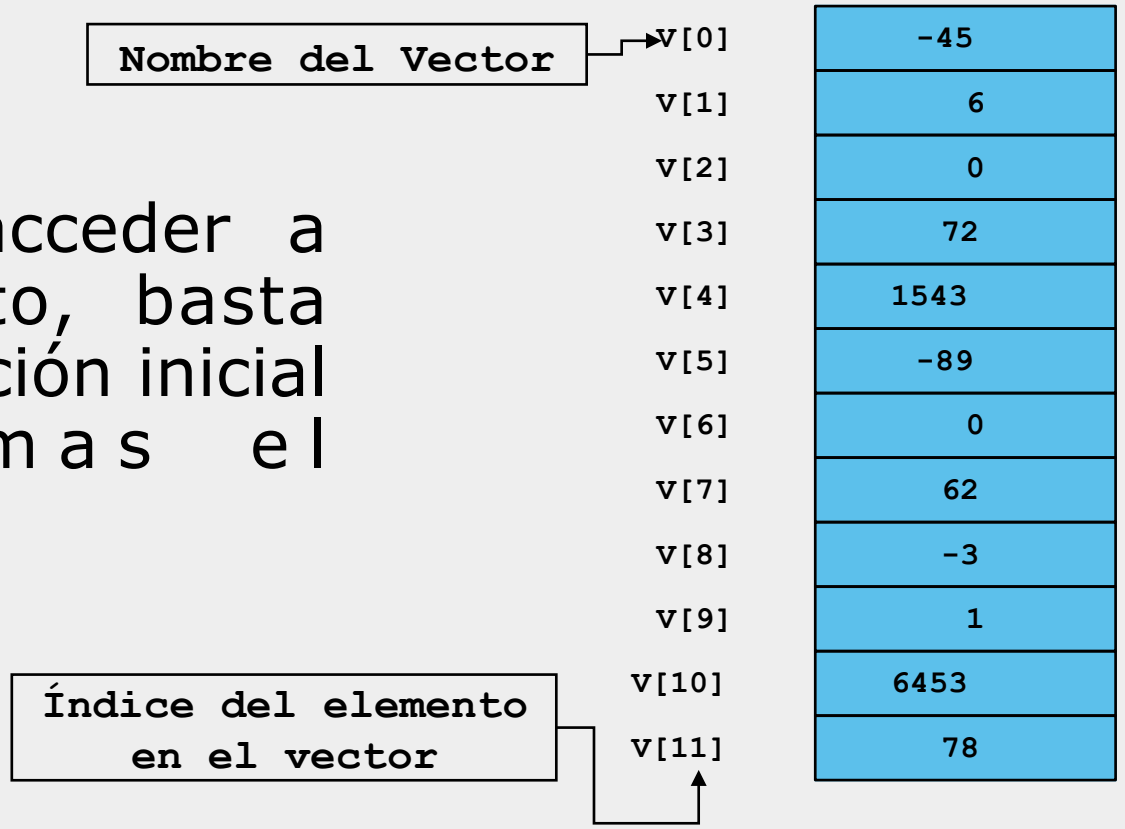
```
#define TAM 100
int main(){
 int v[TAM];
 int i, contador, elemento;

 /*supongamos que v ya tiene valores*/
 printf("Ingrese el elemento a buscar:”);
 scanf(“%d”,&elemento);


 contador = 0;
 for(i=0; i<TAM; i=i+1)
   if (v[i] == elemento)
     contador = contador + 1;
 printf( “El elemento %d aparece %d veces.\n “, elemento,
   contador);
 return 0;
}
```

¿Cómo calcularías el número de elementos diferentes de un vector?

Vectores como Parámetros

La declaración de un vector produce la reserva de un conjunto consecutivo de posiciones de memoria (tantas como componentes tenga el vector).

Notar que para acceder a cualquier elemento, basta con conocer la posición inicial del vector mas el desplazamiento.



Vectores como Parámetros

En el parámetro actual, indicaremos el nombre del vector sin los corchetes.

El paso de un vector

int Medidas[24];

a una función denominada *Calculo*, será similar a

Calculo(Medidas, 24);

Por lo general, siempre se pasa el tamaño del vector como parámetro

Vectores como Parámetros

Los vectores se pasan por referencia (aunque no se indique &)

- El nombre del vector es la dirección del primer elemento
- La función “conoce” donde está almacenado el vector
- La función modifica las posiciones de memoria originales

Por defecto, los valores individuales del vector, son pasados por valor

Vectores como Parámetros

El prototipo de la función será:

void Calculo(int v[], int nroElems);

Como antes, los nombres son opcionales en el prototipo

int v[] puede ser int []

int nroElems puede ser int

Ejemplo

```
/*multiplica por 2 cada
componente del vector*/
#include <stdio.h>
#include <stdlib.h>
#define TAM 12
void doble(int[], int);
void mostrarVector(int[], int);
int main(){
 int vector[TAM];
 int i;
 for(i=0; i < TAM; i=i+1)
    vector[i] = i;
 mostrarVector(vector, TAM);
 doble(vector, TAM);
 mostrarVector(vector, TAM);
 system("PAUSE");
 return(0);
}
```

```
void doble(int T[], int N){
  int i;
  for(i=0; i < N; i=i+1)
       T[i] = T[i] * 2;
   }
```

```
void mostrarVector(int T[], int N){
    int i;
    for(i=0; i < N; i=i+1)
      printf("%d ",T[i]);
    printf("\n");
   }
```

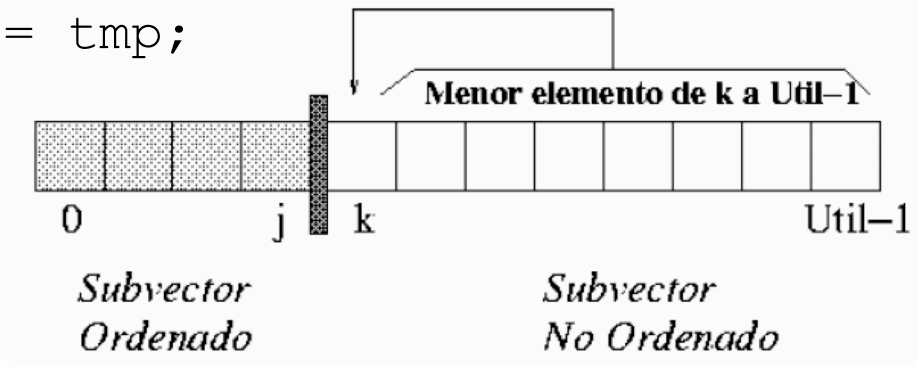
Ejercicios

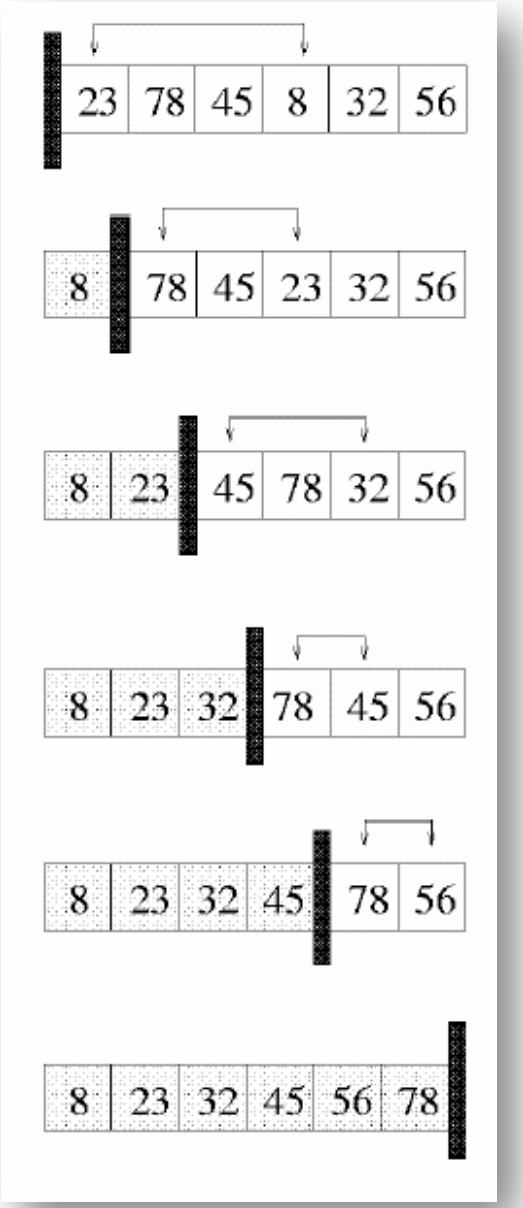
- Dado un vector V de enteros, con elementos *V[i], i ∈ [0,N]*, construir un vector *A* de forma tal que $A[i] = \sum_{j=0}^{i} V[j]$
- Dado un vector C de tamaño N, donde cada componente C[i] representa el i-esimo coeficiente de un polinomio de grado N, construir una función que permita evaluar el polinomio para un valor X dado.
- Dados dos vectores X e Y de enteros, ambos de tamaño N, implemente una función que permita calcular el producto escalar

$$\sum_{i=0}^{N-1} x[i] * y[i]$$

Ordenación de un Vector por Selección

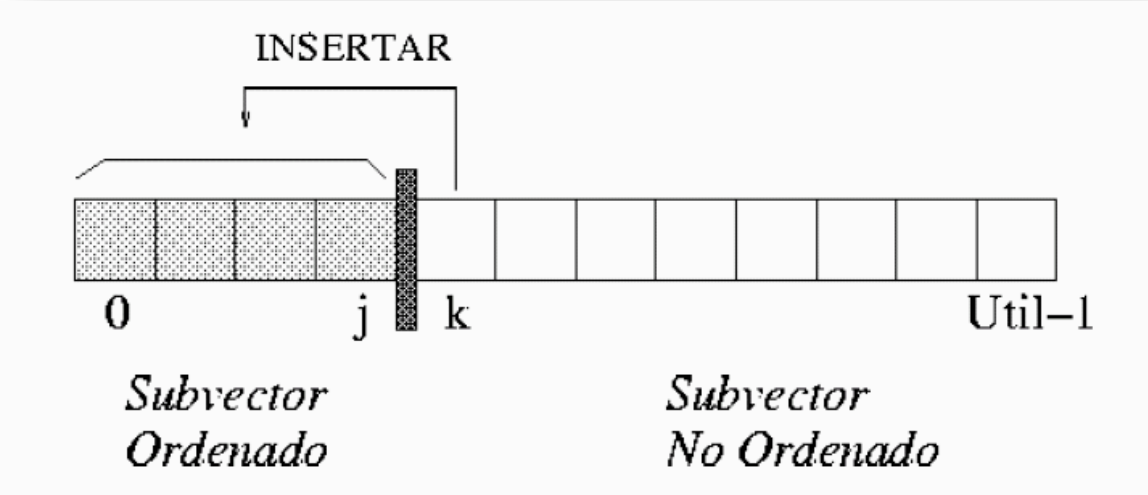
```
void OrdenarSeleccion (double v[], int N){
 int i, j, pos_min;
 double tmp;
 for (i=0; i<N-1; i++) {
  /*Menor elemento del vector v[i..N-1]*/
  pos_min = i;
  for (j=i+1; j<N; j++)
   if (v[j]<v[pos_min])
      pos_min = j;
  /*Coloca el mínimo en v[i]*/
  tmp = v[i];
  v[i] = v[pos_min];
  v[pos_min] = tmp;
 }
}
```

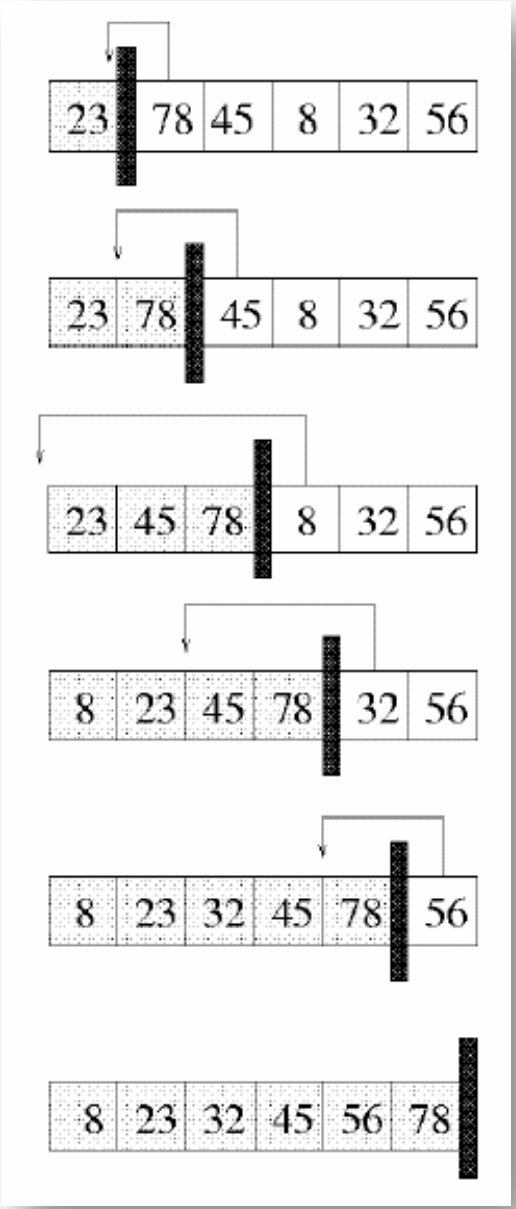




Ordenación de un Vector por Inserción

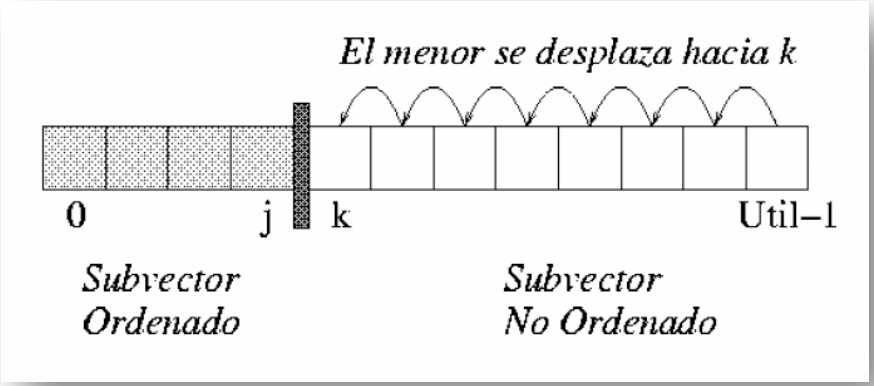
```
void OrdenarInsercion (double v[], int N){
 int i, j;
 double tmp;
 for (i=1; i<N; i++) {
  tmp = v[i];
  for (j=i; (j>0) && (tmp<v[j-1]); j--)
    v[j] = v[j-1];
  v[j] = tmp;
 }
}
```

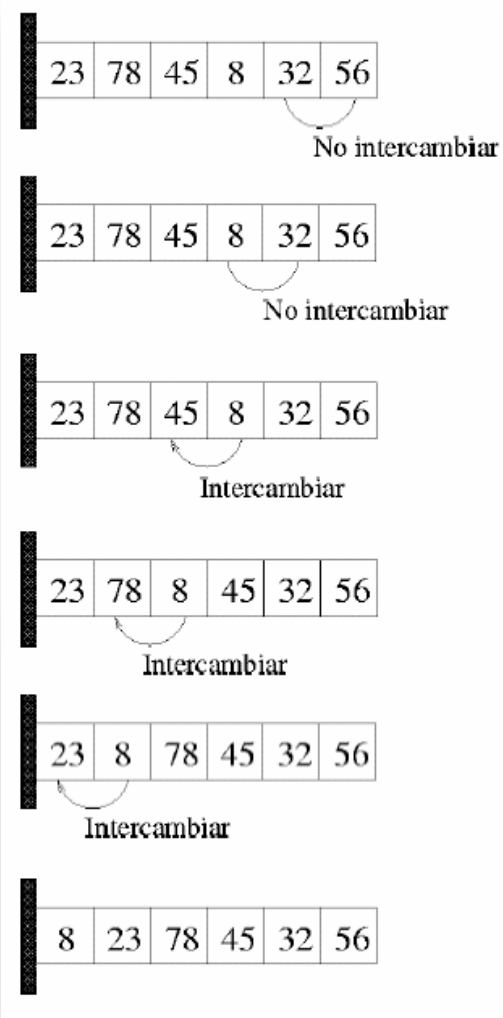




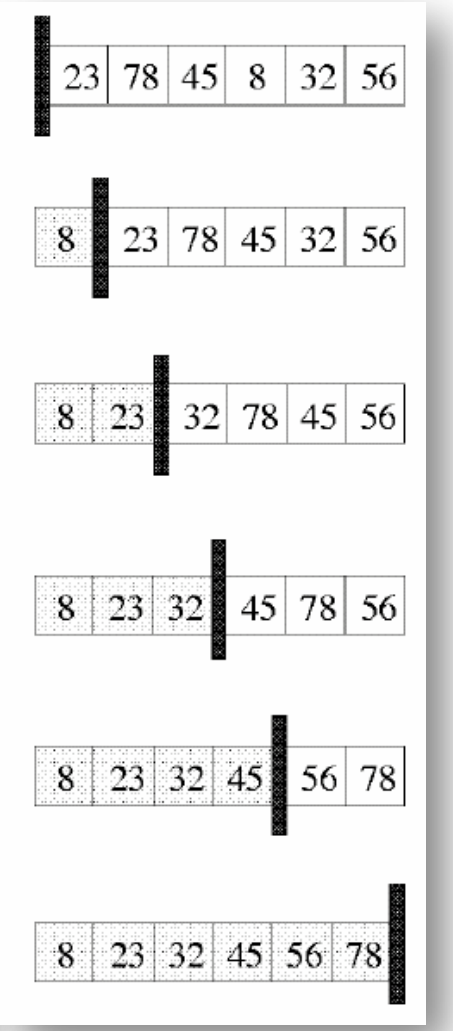
Ordenación de un Vector por Intercambio directo (Burbuja)

```
void OrdenarBurbuja (double v[], int N){
   int i, j;
   double tmp;
   for (i=1; i<N; i++)
      for (j=N-1; j>i; j--)
         if (v[j] < v[j-1]) {
            tmp = v[j];
            v[j] = v[j-1];
            v[j-1] = tmp;
         }
}
```





Primera Iteración



En cada Iteración

Ordenación de un Vector por Ordenación Rápida (Quick Sort)

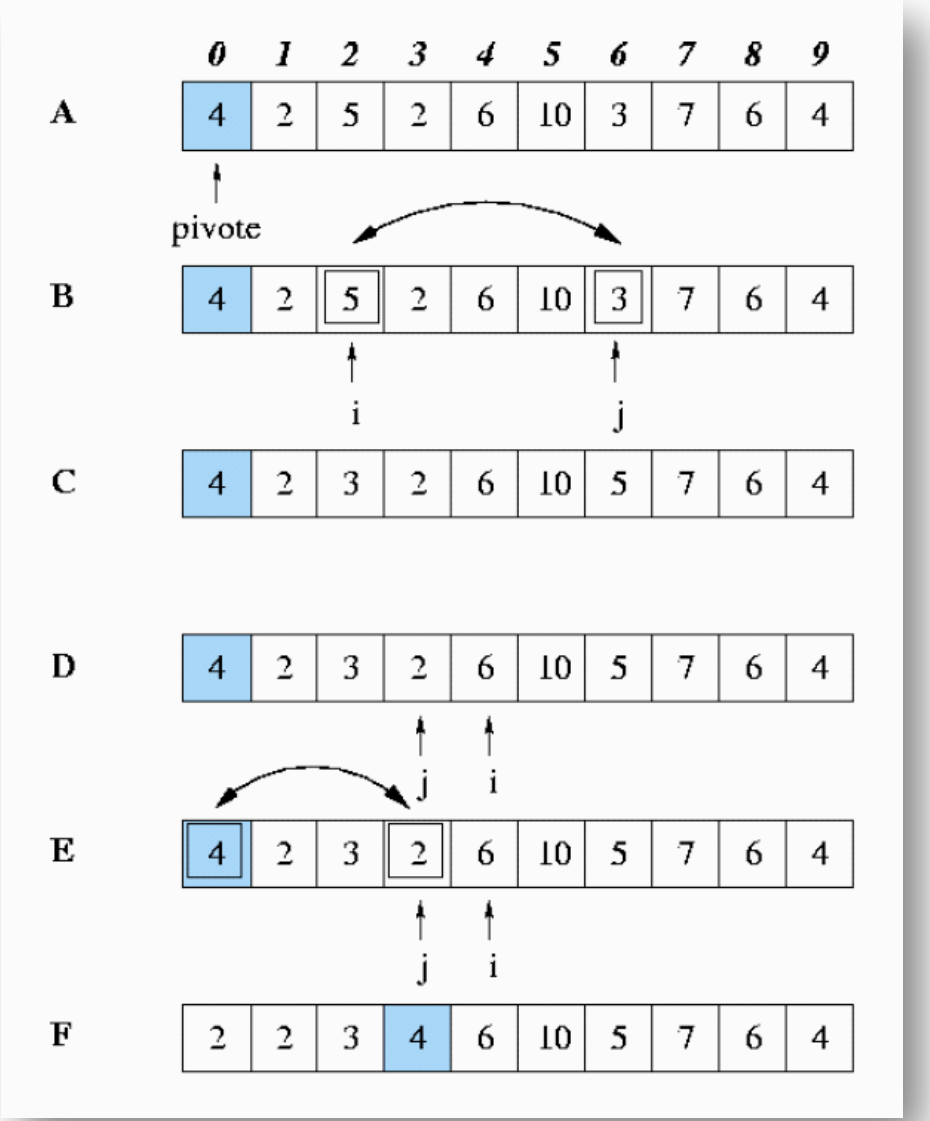
- Se toma un elemento arbitrario del vector, al que denominaremos pivote (p).
- Se divide el vector de tal forma que todos los elementos a la izquierda del pivote sean menores que él, mientras que los que quedan a la derecha son mayores que él.
- Ordenamos, por separado, las dos zonas delimitadas por el pivote.

```
void quicksort (double v[], int izda, int dcha) {
  int pivote; /*posición del pivote*/
  if (izda<dcha) {
   pivote = partir (v, izda, dcha);
   quicksort (v, izda, pivote-1);
   quicksort (v, pivote+1, dcha);
  }
}
```

Obtención del pivote en Quick Sort

Mientras queden elementos mal colocados respecto al pivote:

- Se recorre el vector, de izquierda a derecha, hasta encontrar un elemento situado en una posición i tal que v[i] > p.
- Se recorre el vector, de derecha a izquierda, hasta encontrar otro elemento situado en una posición j tal que v[j] < p.
- Se intercambian los elementos situados en las casillas i y j (de modo que, ahora, v[i] < p < v[j]).



Quick Sort (continuación)

```
/* División del vector en dos partes
   - Devuelve la posición del pivote*/
int partir (double v[], int primero, int ultimo){
  double pivote = v[primero]; /*Valor del pivote*/
  int izda = primero+1;
  int dcha = ultimo;
  do { /*Pivotear…*/
        while ((izda<=dcha) && (v[izda]<=pivote))
           izda++;
        while ((izda<=dcha) && (v[dcha]>pivote))
          dcha--;
      if (izda < dcha) {
         swap ( &(v[izda]), &(v[dcha]) );
        dcha--;
        izda++;
      }
  } while (izda <= dcha);
  /*Colocar el pivote en su sitio*/
  swap (&(v[primero]), &(v[dcha]) );
  return dcha; /*Posición del pivote*/
}
```

```
/*Intercambio de dos valores*/
void swap (double *a, double *b){
  double tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}
```

Matrices: definición y uso

La definición de una estructura multidimensional también es posible.

<tipo><identificador>[d1] [d2]…[dn];

Solo debemos indicar los valores de cada dimensión. Por ejemplo

int a[3][4];

define una estructura *"a"* de 3 filas y 4 columnas.

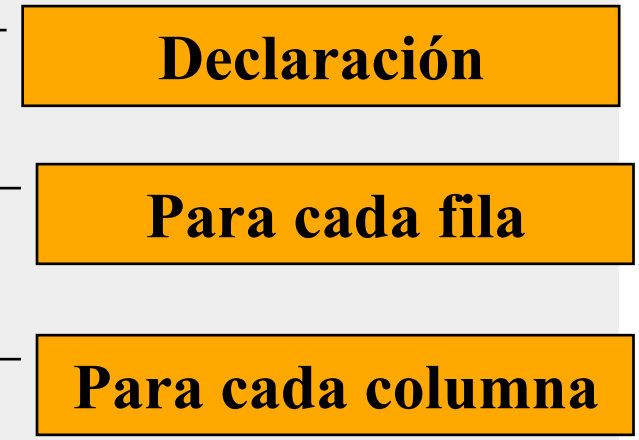
a[0][0]	-45
a[0][1]	6
a[0][2]	0
a[0][3]	72
a[1][0]	1543
a[1][1]	-89
a[1][2]	0
a[1][3]	62
a[2][0]	-3
a[2][1]	1
a[2][2]	6453
.....	78

	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Fila 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Fila 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Matrices: definición y uso

El acceso y manipulación de componentes sigue el mismo formato que en los vectores.

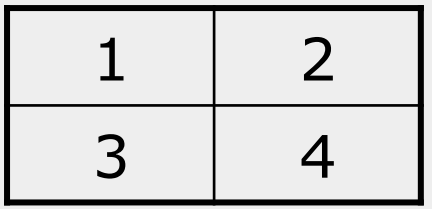
```
#define FILA 10;
#define COL 20;

int main(){
   int mat[FILA][COL];
   int i,j;
   for(i=0; i < FILA; i=i+1){
      for(j=0; j < COL; j=j+1)
            manipulo mat[i][j];
   }
}
```



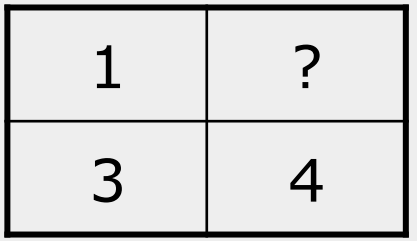
Matrices: definición y uso

Inicialización

```
int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
```




```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```



Paso de Matrices a Funciones

En el parámetro actual, indicaremos el nombre de la matriz sin los corchetes.

El paso de una matriz

int tabla[24][12];

a una función denominada *Calculo*, será similar a

Calculo(tabla, 24,12);

Por lo general, siempre se pasan todos los valores de las dimensiones

Paso de Matrices a Funciones

El prototipo de la función será:

void Calculo(int V[][12], int nroFil, int nroCol);

Si la matriz tiene mas de dos dimensiones, entonces en el prototipo deben indicarse los valores máximos para todas excepto la primera.

void tresD(int v[][10][15], int d1, int d2, int d3);

Ejemplo

```
#include <stdio.h>
#define FILA 4
#define COL 8
void cargaMatriz(int M[][COL],
                 int f, int c);
int main(){
  int mat[FILA][COL];
  int i,j;
  cargaMatriz(mat, FILA, COL);
  for(i=0; i < FILA; i=i+1){
    printf("\n");
    for(j=0; j < COL; j=j+1)
      printf("%d \t", mat[i][j]);
  }
}
```

```
void cargaMatriz(int M[][COL],
                 int f, int c){
  int i,j;
  for(i=0; i < f; i=i+1){
    for(j=0; j < c; j=j+1)
        M[i][j] = i*COL + j;
  }
}
```

Ejercicio: Implementa una función para obtener la traspuesta de una matriz

Ejercicios

- Dada una matriz de dimensión MxN, utilizando resultados de vectores vistos en este capítulo, ¿cómo ordenarías sus elementos en orden creciente por filas y columnas? (primera fila ordenada, continuando la ordenación con la segunda, tercera, etc..)
- Implementa funciones para sumar, restar y multiplicar matrices introducidas por teclado.