Tema 5:
Estructuras de Datos

Objetivos

Conocer y saber utilizar diferentes tipos de datos estructurados:

- cómo se definen
- cómo están organizadas sus componentes
- cómo se accede a ellas y
- sus operaciones y algoritmos básicos

Índice

- Estructuras de datos en Memoria Principal
  - Vectores
  - Matrices
  - Cadenas de caracteres
  - Estructuras

Estructuras de datos

Tipo de datos estructurado:

- Están formados por otros datos (simples o también estructurados) entre los que hay una relación lógica.

- Un dato estructurado agrupa colecciones de datos que interesa manejar globalmente, p.ej. una fecha

- Se caracterizan por:

  El tipo de las **componentes** que se estructuran

  El método de estructuración

Estructuras de datos

- **Estructuras de datos en Memoria Principal**
  - Estáticas: Arrays, Cadenas, Estructuras…
  - Dinámicas

Estructuras acotadas:	**Estructuras no acotadas:**
El número de componentes **se fija** en el momento de la definición.	El número de componentes **puede variar** durante la ejecución.

- **Estructuras de datos en Memoria Externa**
  - Ficheros

Arrays

- Un **array** es una colección de datos todos del mismo tipo, a las que se llama con un mismo nombre.
- Los elementos de un array se referencian a través de un nombre y un índice. El nombre es común a los elementos y el índice indica la posición del elemento (componente) dentro del array.
- El número de componentes de un array es finito y prefijado de antemano (estructura acotada).
- Las componentes de un array se almacenan en memoria en posiciones consecutivas.

Arrays

- Cuando el array tiene una única dimensión se le llama **vector.**

- Los datos almacenados en un array se llaman **elementos** y se **numeran** consecutivamente del **0 en adelante**. A estos números se les llama **índices** del array.

- El tipo de datos de los elementos puede ser cualquier tipo de datos de C.

	0	**1**	**2**	**3**	**4**
notas	5.0	3.2	7.1	5.6	8.8

Declaración de Arrays

- Para declarar un array debemos indicar el **tipo de datos** de sus elementos y el **tamaño** o longitud del array, es decir, cuántos elementos tiene:

tipo_elem nombre_array [tamaño]

Siendo:

tipo_elem: el tipo de los elementos del vector

nombre_array: identificador

tamaño: indica el número de elementos del vector, puede ser una expresión constante

Declaración de Arrays. Ejemplos

- Ejemplo 1:

```
float  notas[5]       //define un array de 5 elementos reales
float  alturas[20]    //define un array de 20 elementos reales
int  edades[10]       //define un array de 10 elementos enteros
char  linea[80]       //define un array de 80 elementos carácter
```

- Ejemplo 2:

```
#define Max  4
 void main(void)
{
   int vector[Max],  v2[Max+1];
   ...
```

Acceso a los Elementos

- Para acceder a cada uno de los elementos del array:

nombre_array[posición]

Siendo:

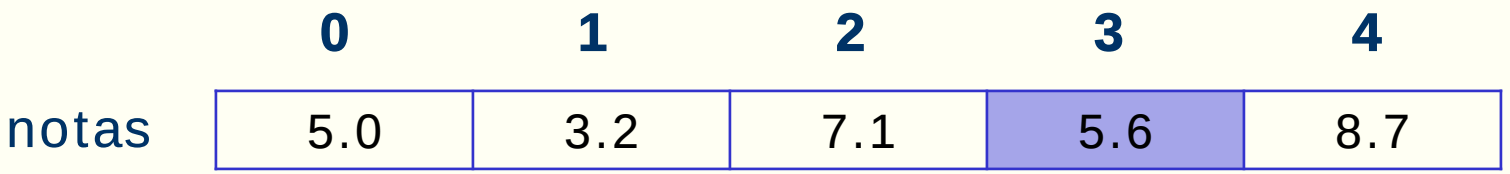
posición: una expresión de tipo entera. Es un valor comprendido entre 0 y (tamaño del vector) – 1

Acceso a los Elementos. Ejemplos

Si tenemos declarado:

float notas[5]

y hemos asignado valores al vector. Gráficamente :



notas[3] accede a la cuarta componente del array notas

Ejemplos de formas de acceso a las componentes del vector notas:

```
notas[3]=5.6;

printf("%f", notas[3]);

notas[3]=notas[3]+1;
```

La posición se puede poner también como una expresión entera:

```
int  i = 2;
   notas[i]  = notas[i] * 3;
   notas[i+2] = 8.7;
```

Arrays

- C **no comprueba los límites** del array, es responsabilidad del programador no rebasar los límites.

  Ejemplo:

  Si tenemos declarado el vector:

  ```
  int  v[5];
  ```

  El compilador no da error si hacemos:

  ```
  v[7] = 34;     //pero es un error  puede existir sobreescritura
  ```

- **Operaciones y restricciones:**
  - **No** se pueden comparar vectores, ni asignar directamente un vector en otro, ni lectura o escritura directa.
  - Estas operaciones se efectúan **elemento a elemento**.

Inicialización de un Array

- Existen varias formas de inicializar los elementos de un array:
  - con **asignaciones**, recorriendo el vector
  - con **lecturas**, sobre cada uno de los elementos
  - en la **declaración**, asignando valores

Inicialización de un Array

- con **asignaciones**, recorriendo el vector:

```
float notas[5];
int i;
for (i=0; i<5; i++)
  notas[i] = 0.0;
```

- con **lecturas**, sobre cada uno de los elementos:

```
float notas[5];
int i;
printf (“\n Escribe las 5 notas: ”);
for (i=0; i<5; i++)
  scanf(“%f”,&notas[i]);
```

Inicialización de un Array

- en la **declaración**, asignando valores:

```
float notas[5]= {10.0, 8.5, 3.75, 7.0, 7.5};
int v[3]= {5, 16, -7};
int vect[8] ={0};      // almacena 0 en las ocho componentes
```

De otra forma: se pueden dejar los **corchetes vacíos**, hay que introducir tantos valores como elementos tenga el vector

```
float notas[]= {10.0, 8.5, 3.75, 7.0, 7.5};
int vector[]= {5, 16, -7}
```

Ejemplo

Realizar un programa C para almacenar las 15 notas de un examen, calcular la nota media y mostrar en pantalla cuántas notas están por encima de la media.

```
#include <stdio.h>
#define  N  15
void main()
{
      float notas[N], media=0.0;
      Int  i, cont=0;

      for(i=0; i<N; i++)
      {      printf("intro un numero para notas[%d] ",i);
             scanf("%f", &notas[i]);
             media=media+notas[i];
      }
      media=media/N;
       for(i=0; i<N; i++)
             if(notas[i]>media)  cont++;
      printf("\nHay %d notas por encima de la nota media: %.2f", cont, media);
}
```

Arrays como parámetros de función

- En C **no se puede pasar el array completo** como parámetro de una función.

- La solución es pasar la **dirección de memoria** donde comienza a almacenarse el array. **Este valor se representa por el nombre del array**. Es decir, **los arrays** son siempre **parámetros variables** (de salida o de entrada/salida).

- En el lenguaje C el **paso de arrays** a las funciones es una **excepción** al convenio de llamada por valor estándar.

- En la **llamada a la función,** el lenguaje C, trata al argumento (parámetro real o actual), como si hubiera situado el operador & delante del nombre del array. El nombre el array es un puntero al primer elemento de vector.

Arrays como parámetros de función

- La **declaración**, en una función, **de un parámetro** tipo vector se realiza con uno de los siguientes formatos:

tipo_elem nombre_array [tamaño]

tipo_elem nombre_array []

tipo_elem *nomb

- En la llamada a una función: para pasar un array como **parámetro**, hay que especificar el **nombre del array sin corchetes.** Llamada a la función:

nombre_func (nombre_array)

El nombre del array es la **dirección del primer elemento del array** (no se pone & delante del nombre del vector).

Arrays como parámetros de función

Ejemplo:

función que calcule la media de los elementos de un vector.

- la cabecera de la función mediaNotas, podría ser:

```
float  mediaNotas(float n[20])
```

o bien:

```
float  mediaNotas(float n[])
```

- Si desde main queremos escribir la media de las notas con una **llamada a la función** mediaNotas, pondríamos:

```
...
void main() {
  float notas[20];
  ...
  printf("%f", mediaNotas(notas));
```

/*notas debe aparecer sin [ ] y sin &*/

Ejemplo

- Función que suma los elementos de un vector.

```
#include <stdio.h>
int sumar (int v[]);
void main ()
{
        int i;
        int vector[10]={10,11,12,13,14,15,16,17,18,19};
        for (i=0; i<10; i++)
           printf("%d  ",vector[i]);
     printf("\n\nSuma de los elementos del array: ");
        printf("%3d",sumar(vector));
}

int sumar (int v[10])
{
        int result=0, j;
        for (j=0; j<10; j++)  result = result + v[j];
        return result;
}
```

Vectores. Operaciones básicas

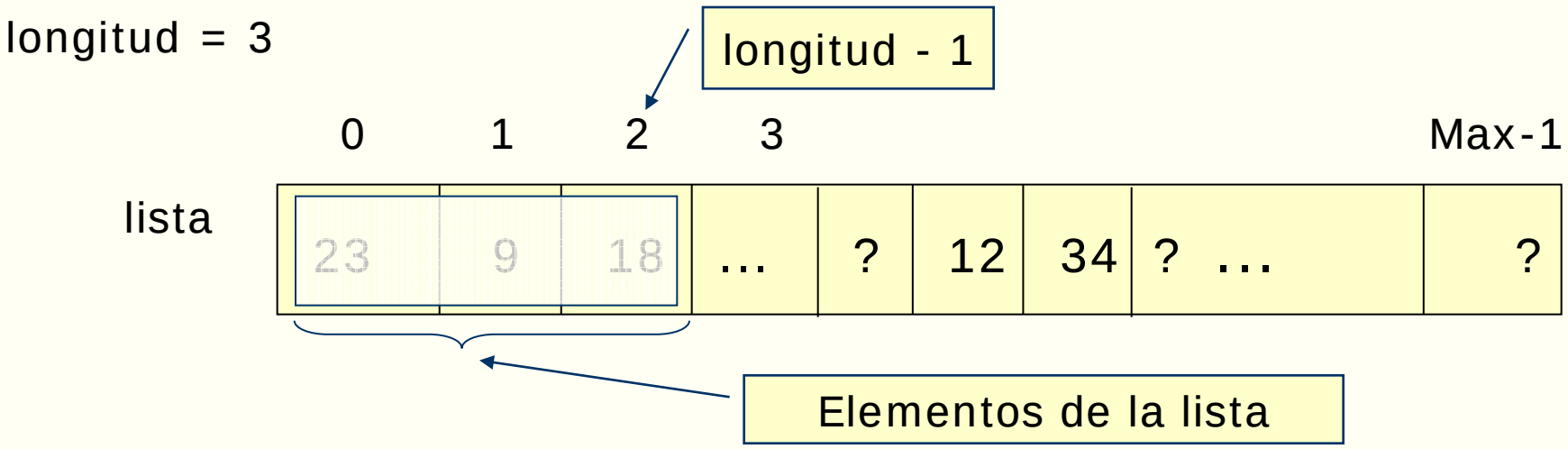
- **Recorrer** las componentes**:**
  - **Inicializar el vector**
  - **Leer** en las componentes
  - **Escribir** las componentes
- **Insertar** o **eliminar** un elemento
- **Buscar** un elemento
- **Ordenar** las componentes

Vectores. Listas

REPRESENTAR una lista (subarray) de máximo 100 elementos enteros:

```
#define Max  100
 void main(void)
 {
    int   lista[Max], longitud;
    // longitud contiene la cantidad de elementos válidos en el vector
```

Gráficamente, después de dar valores a las componentes:



Operaciones con Arrays

- **Recorrer un Vector.** Pasa por todos los elementos del vector desde el primero (posición 0) hasta el último realizando la operación deseada: escribir el contenido, almacenar un valor, etc.

  Ejm: escribir las componentes de un vector que tiene como máximo 10 elementos junto a su posición

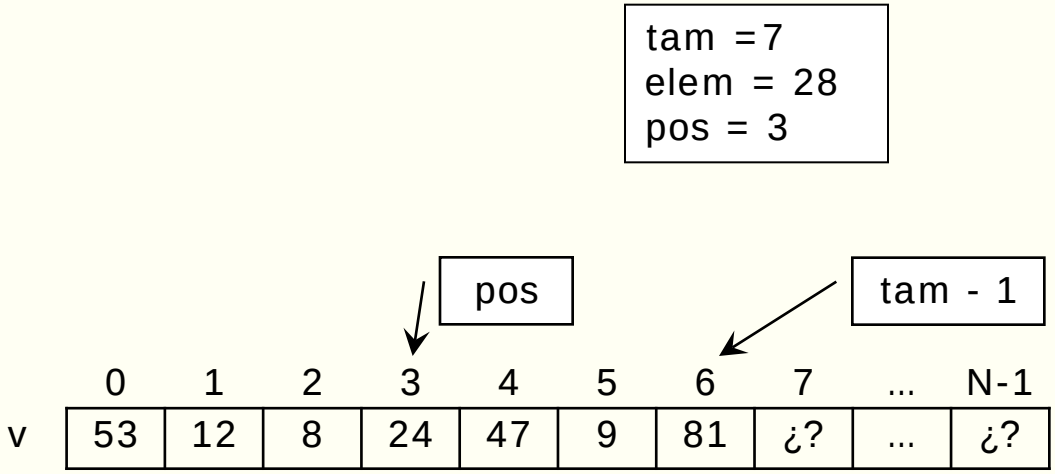
```
/*Se incluye el parámetro tam para indicar el tamaño del subarray
   con que vamos a trabajar */

void Recorrido (int V[10], int tam)
{
  int i;
  for (i=0; i< tam; i++)
    printf("el elemento de la posición %d es: %d", i, V[i]);
}
```

Operaciones con Arrays

- **Insertar un elemento.** Ejemplo. Situación de partida:

  Llamamos a la función con los siguientes valores:



Cabecera de la función:

void Insertar(int V[N], int elem, int pos, int *tam)

//inserta en la componente pos de V el valor elem

//si el vector está lleno pierde el último valor

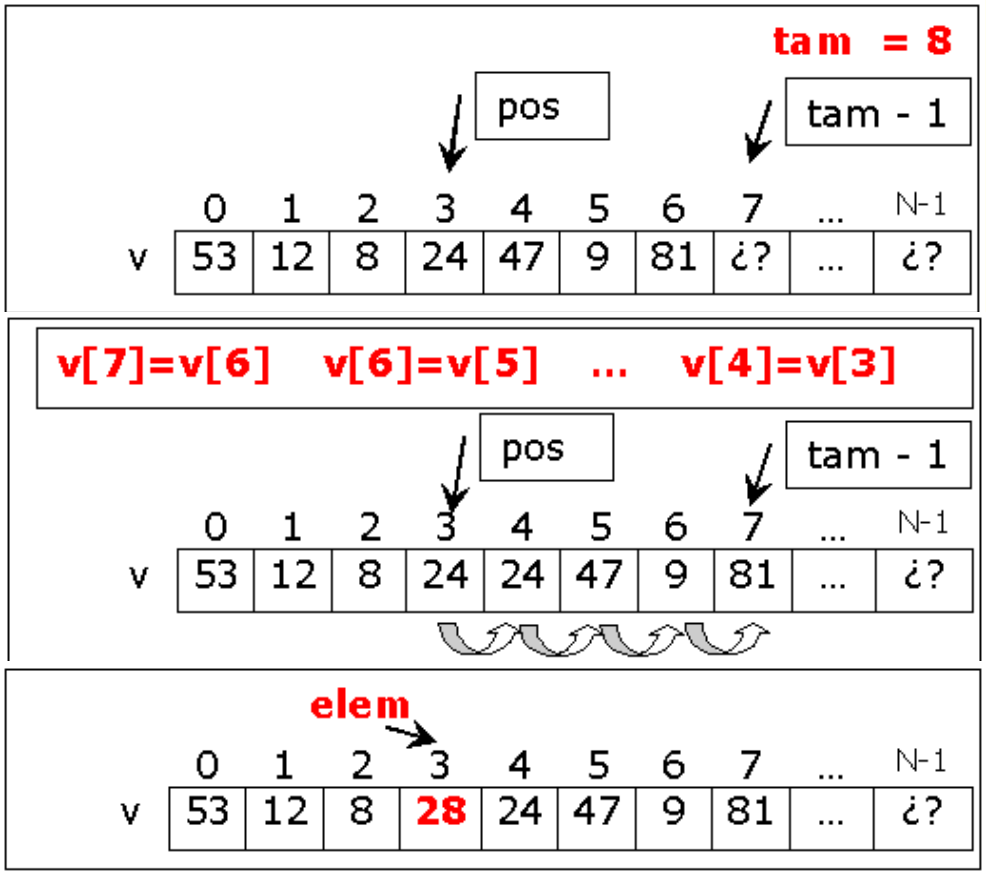
Operaciones con Arrays

- **Insertar un elemento.** Introduce el nuevo elemento en la posición que se indica, desplazando los elementos desde dicha posición una posición a la derecha. Si el vector está lleno, se pierde el último elemento del vector.

```
void Insertar(int V[N], int elem, int pos, int *tam)
{  //inserta en la componente
   // pos de V el valor elem
   //si el vector está lleno
   // pierde el último valor




}
```



Operaciones con Arrays

- **Borrar un elemento.** Elimina el elemento de la posición que se indica, desplazando los elementos que se encuentran a su derecha una posición a la izquierda, y se disminuye el número de elementos almacenados en el vector en una unidad.

```
void  Borrar(int V[N], int pos, int *tam)
{
   //borra de V  el elemento de índice pos



   for(i=pos;                    )
      V[i]= V[    ];


}
```

Vectores. Búsqueda

OPERACIONES con Vectores:

Búsqueda: consiste en examinar una colección de datos para determinar si contiene o no un elemento específico.

Entre los distintos algoritmos de búsqueda vamos a estudiar:

búsqueda secuencial y dicotómica (o binaria).

Operaciones con Arrays

- **Búsqueda Secuencial:** Busca un elemento, recorriendo secuencialmente el array, comenzando en la primera posición del array y se detiene cuando encuentra el elemento buscado o bien se alcanza el final del array.

  El algoritmo comprueba el elemento almacenado en la primera posición, a continuación el segundo y así sucesivamente hasta que encuentra el elemento buscado o se termina el recorrido.

Búsqueda Secuencial

```
int BusquedaSec (int V[N], int elem, int tam)
{
   int i=0, enc=0;

   while ((!enc) && (    ))
   {
      if (V[i]==elem)

      else

   }
   if (!enc)


   return          ;
}
```

Operaciones con Arrays

- **Busqueda Dicotómica o Binaria:** Si el array está ordenado, la búsqueda binaria proporciona un método de búsqueda mejorada.

  Se sitúa la búsqueda en el centro del array y se comprueba si el elemento buscado coincide con el valor del elemento central. Si no coincide, se sitúa uno en la mitad inferior o superior del elemento central del array para seguir buscando.

Búsqueda Binaria

```
int BusquedaBin (int V[N], int elem)
{
    int ini, med, fin, pos   ;
    ini=0;
    fin = N-1;


    while (          )
    {




    }


   return      ;
}
```

Operaciones con Arrays. Ordenación

Ordenación: hay muchos algoritmos desarrollados debido a su importancia en diferentes aplicaciones:

- La ordenación consiste en colocar una serie de datos en orden ascendente o descendente.
- La elección de un determinado algoritmo depende de la ubicación , la cantidad, del tipo de datos ...

Ordenación

Algoritmo de ordenación por intercambio o de burbuja:

se recorre varias veces el vector hasta colocar en su sitio todos los elementos:

- En el primer recorrido del vector (coloca el elemento mayor en su lugar correcto):
  - compara A[0] y A [1], si están en orden, se mantienen como están, en caso contrario, se intercambian entre sí.
  - A continuación se comparan los elementos A [1] y A [2], de nuevo se intercambian si es necesario.
  - El proceso continua hasta comparar los dos últimos elementos del vector.
- En cada uno de los siguientes recorridos coloca en el lugar correcto un valor más:
  - Se siguen los mismos pasos anteriores pero cada vez haciendo una comparación menos.

Ordenación

Algoritmo de intercambio o de burbuja:

Ejemplo:

	0	1	2	3	4
A	40	15	12	55	1

Primer recorrido del vector: el número mayor 55 se quedará en su sitio

		0	1	2	3	4
A[1] > A [2] y los intercambia:	A	15	40	12	55	1
A[2] > A [3] y los intercambia:	A	15	12	40	55	1
A[3] < A [4] los deja igual:	A	15	12	40	55	1
A[4] > A [5] y los intercambia:	A	15	12	40	1	55

Ordenación

Segundo recorrido del vector : el número 40 se quedará en su sitio

		0	1	2	3	4
A[1] > A [2] los intercambia:	A	12	15	40	1	55
A[2] < A [3],	A	12	15	40	1	55
A[3] > A [4], los intercambia:	A	12	15	1	40	55

Tercer recorrido del vector : el número 15 se quedará en su sitio

		0	1	2	3	4
A[1] < A [2]	A	12	15	1	40	55
A[2] > A [3] los intercambia:	A	12	1	15	40	55

Cuarto recorrido del vector : el vector se quedará ordenado

		0	1	2	3	4
A[1] > A [2] los intercambia:	A	1	12	15	40	55

Ordenación. Selección directa

- Algoritmo de ordenación por selección directa.

  se repite el siguiente proceso desde el primer elemento hasta el penúltimo (componente tratada):

  – se selecciona la componente de menor valor de todas las situadas a la derecha de la tratada

  – se intercambia el elemento buscado con la componente tratada

Ordenación. Selección directa

- **Algoritmo de ordenación por selección directa. Ejemplo:**

Secuencia inicial: 9 4 7 1 4

Primera búsqueda: (9) 4 7 [1] 4

1 4 7 9 4

Segunda búsqueda: 1 (4) 7 9 4

Tercera búsqueda : 1 4 (7) 9 [4]

1 4 4 9 7

Cuarta Búsqueda: 1 4 4 (9) [7]

1 4 4 7 9

(○) **Componente tratada** [□] **Componente menor**

Ordenación. Selección directa

```
void OrdSelecc (int V[N])
{
    int imenor, i, j, aux;
    for (i=0; i<N-1; i++)
    {




    }
}
```

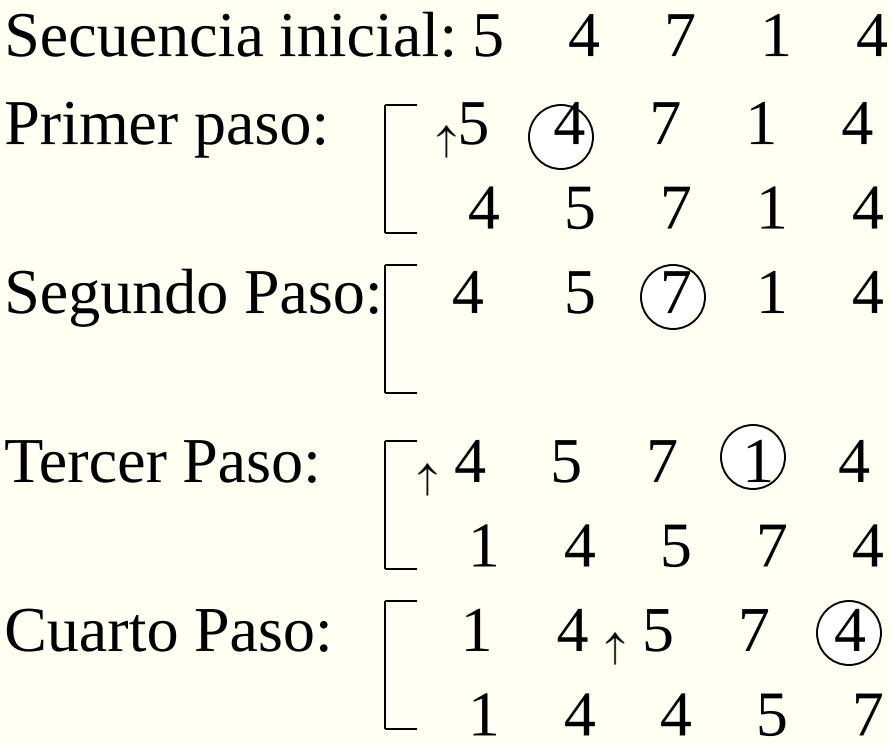
Ordenación. Inserción directa

- **Algoritmo de ordenación por inserción directa:**
  se toman todos los elementos desde el segundo hasta el último y con cada uno de ellos se repiten las siguientes operaciones:
  - se copia el elemento en una variable auxiliar
  - desde el anterior al que tratamos hasta el primero, desplazamos un lugar a la derecha los que sean mayores que el tratado para buscar su hueco
  - se inserta el elemento en el hueco

Ordenación. Inserción directa

- **Algoritmo de ordenación por inserción directa.**

  Ejemplo:



○ **Componente tratada** ↑ **Hueco**

Ordenación. Inserción directa

```
void OrdInserc(int v[N])
{
   int i, j, aux;
   for(i=1; i<N ; i++)
   {









   }
}
```

Array de dimensión múltiple o matriz

- Se pueden definir arrays de más de una dimensión:

  tipo_elem nombre_matriz [dim1][dim2]...[dimN]

- Ejemplo:

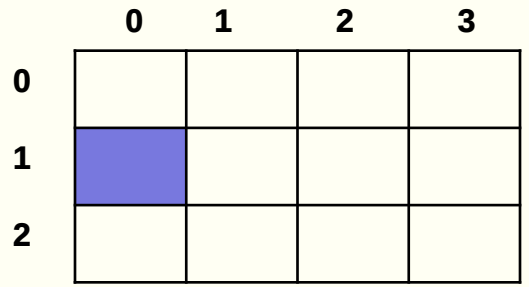
  int matriz [3][4] matriz de dos dimensiones,

  con 3 filas y 4 columnas

- Para acceder a los elementos de la matriz se necesitan tantos índices como dimensiones tenga.

  Ejemplo:

  matriz [1][0] hace referencia al elemento de la segunda fila, primera columna



Inicialización de un Matriz

- Existen varias formas de inicializar los elementos de un matriz:
  - con **asignaciones**, recorriendo el array
  - con **lecturas**, sobre cada uno de los elementos
  - en la **declaración**, asignando valores

Inicialización Matriz

– con **asignaciones**, recorriendo la matriz:

```
int matriz[3][4];
int f,c;
for (f=0; f<3; f++)
  for (c=0; c<4; c++)
       matriz[f][c] = 0;
```

– con **lecturas**, sobre cada uno de los elementos:

```
printf (“\n Escribe los valores: ”);
for (f=0; f<3; f++)
  for (c=0; c<4; c++)
    scanf(“%d”,&matriz[f][c]);
```

Inicialización Matriz

– en la **declaración**, asignando valores:

```
int matriz[3][4]={{0,0,0,0},{0,0,0,0},{0,0,0,0}};
int matriz[3][4]={0}    //equivalente a la anterior
```

```
int matriz[][]={{11,5,7,14},{-6,0,44,9},{56,4,0,10}};
```

```
int matriz[3][4]={11,5,7,14,-6,0,44,9,56,4,0,10};
```

Arrays como parámetros de función

- La declaración en la función de un parámetro tipo matriz:

```
tipo_elem  nomb_array [dim1][dim2]

tipo_elem  nomb_array [][dim2]

tipo_elem  *nomb
```

Ejemplo: la cabecera de la función EscribeMatriz, podría ser:

```
void  EscribeMatriz( int M[3][4])
```

- En la llamada pondremos solo el nombre del array:

Ejemplo:

```
int  mat[3][4];
EscribeMatriz (mat);
```

Ejemplos

- Escribir una función C para crear la matriz identidad de dimensión 5. La matriz identidad tiene todos sus valores a 0 excepto los de la diagonal principal que es 1.

```
void matrizIdent (int identidad [5][5])
{
    int f,c;









}
```