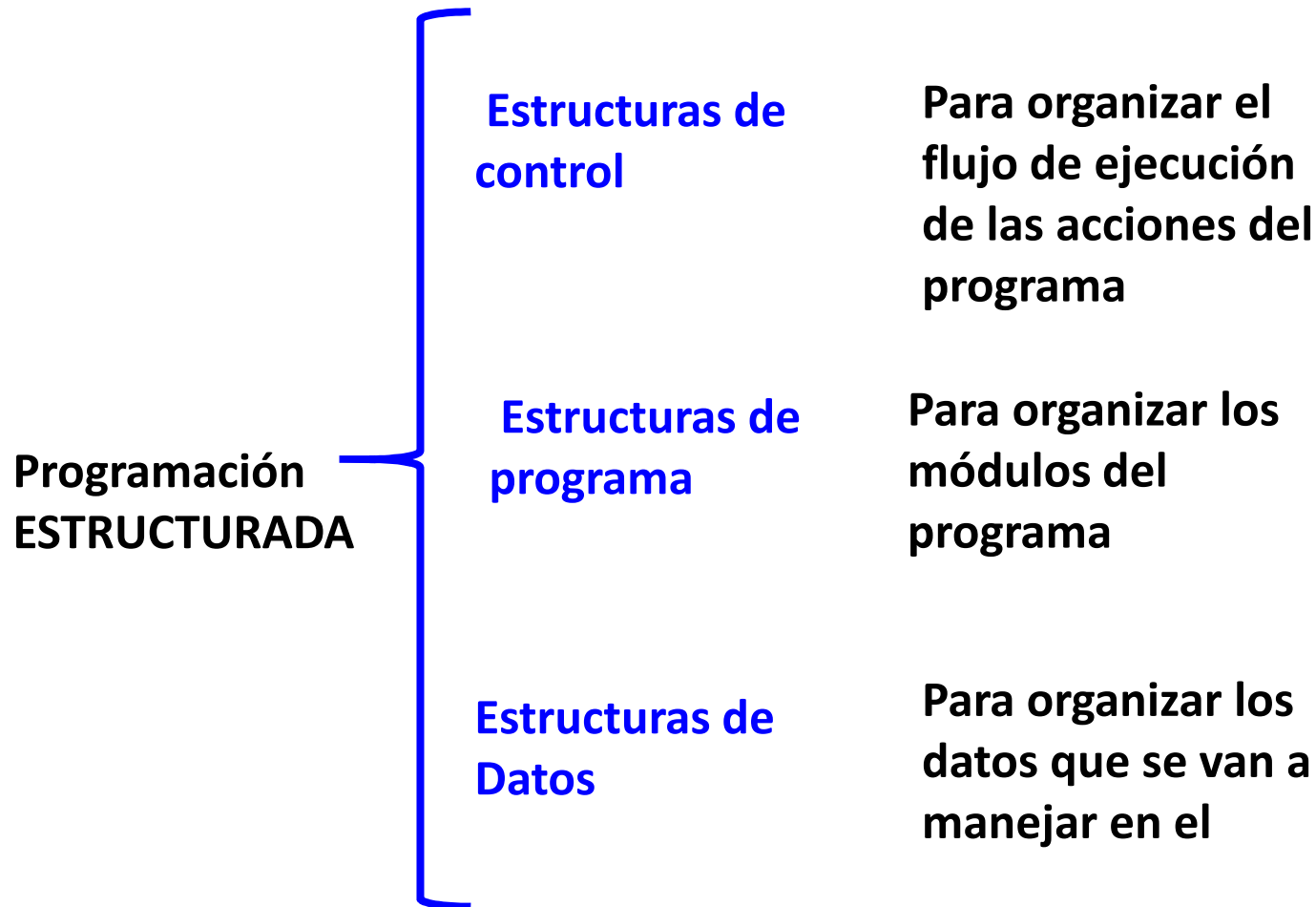


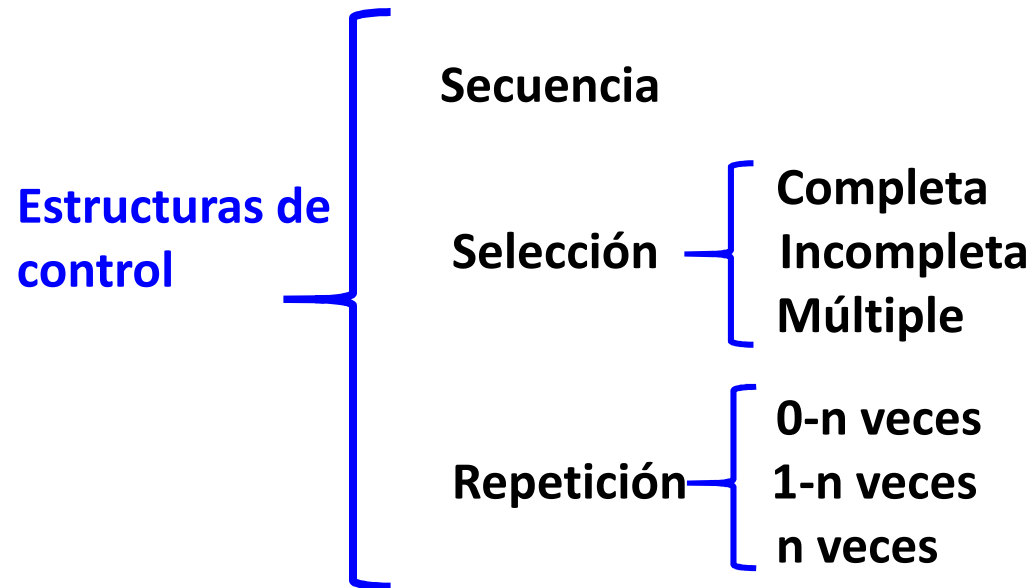
Algoritmos y **E**structuras **D**e **D**atos

ESTRUCTURAS DE DATOS

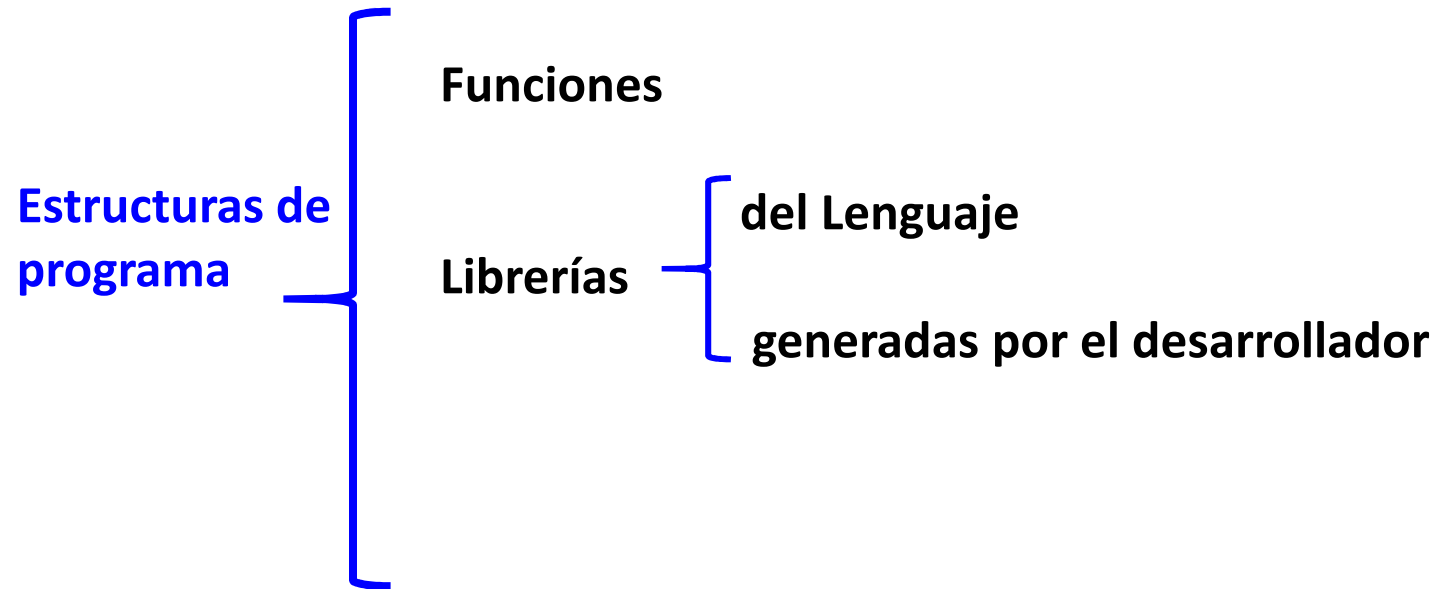
ESTRUCTURAS: de Control, de Programas y de Datos



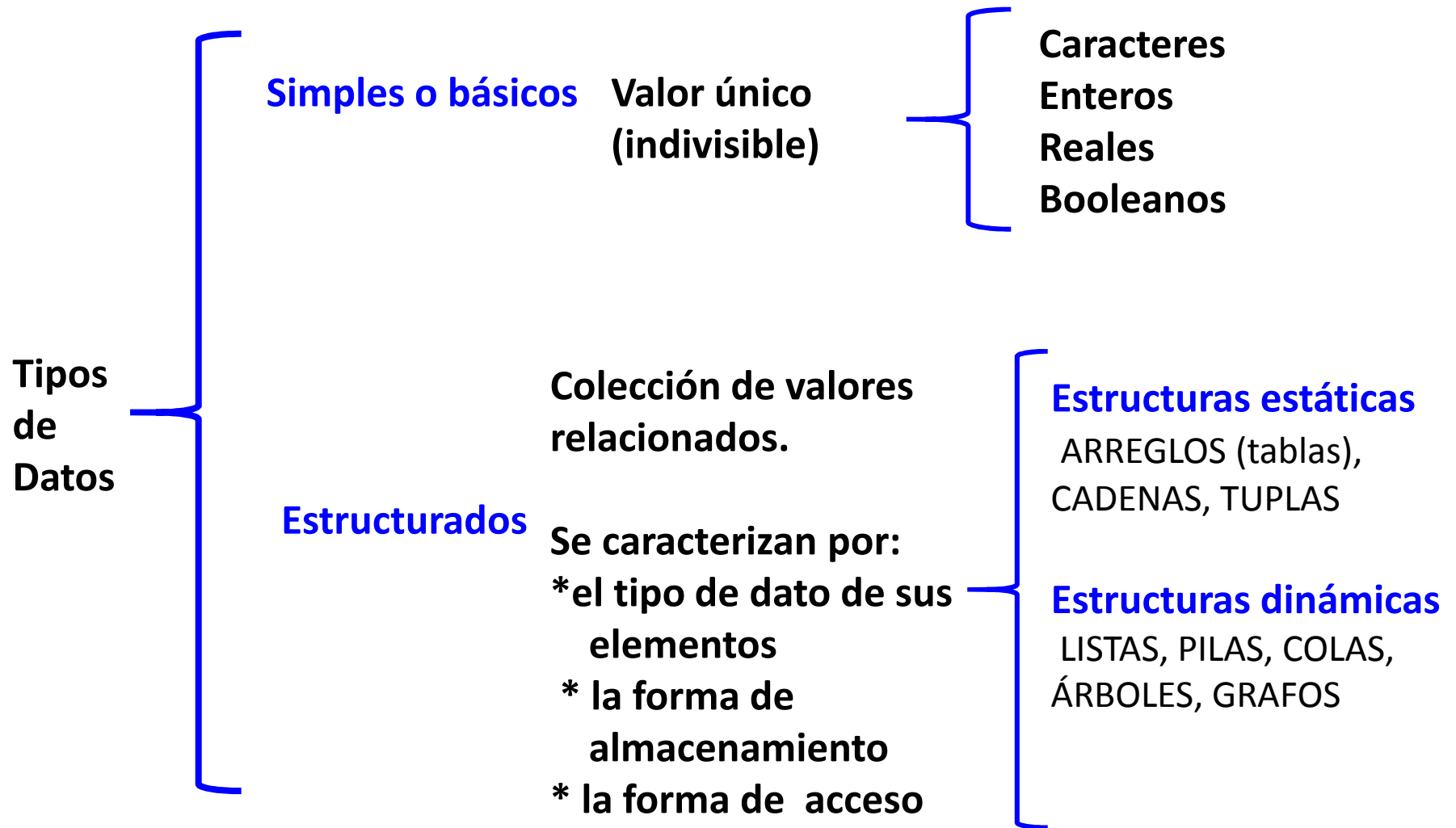
ESTRUCTURAS de Control



ESTRUCTURAS de Programa



TIPOS de datos



ESTRUCTURAS DE DATOS: Clasificaciones

- Según donde se almacenan
 - Internas** (en memoria principal)
 - Externas** (en memoria auxiliar)
- Según tipos de datos de sus componentes
 - Homogéneas** (todas del mismo tipo)
 - No homogéneas** (pueden ser de distinto tipo)
- Según la implementación
 - Provistas por los lenguajes** (básicas)
 - Abstractas** (TDA - Tipo de dato abstracto que puede implementarse de diferentes formas)
- Según la forma de almacenamiento
 - Estáticas** (ocupan posiciones fijas y su tamaño nunca varía durante todo el módulo)
 - Dinámicas** (su tamaño varía durante el módulo y sus posiciones también)

ESTRUCTURAS DE DATOS

- **Simples o básicos:** caracteres, reales, flotantes.
- **Estructurados:** colección de valores relacionados. Se caracterizan por el tipo de dato de sus elementos, la forma de almacenamiento y la forma de acceso.

- **Estructuras estáticas:** Su tamaño en memoria se mantiene inalterable durante la ejecución del programa, y ocupan posiciones fijas.

ARREGLOS - CADENAS - ESTRUCTURAS

- **Estructuras dinámicas:** Su tamaño varía durante el programa y no ocupan posiciones fijas.

LISTAS - PILAS - COLAS - ARBOLES - GRAFOS

Tipos Compuestos

Los tipos compuestos surgen de la composición y/o agregación de otros tipos para formar **nuevos tipos de mayor entidad**.

Existen **dos formas fundamentales de crear tipos de mayor entidad**:

→ La **composición de elementos**, que denominaremos “**Registros**” o “**Estructuras**” y

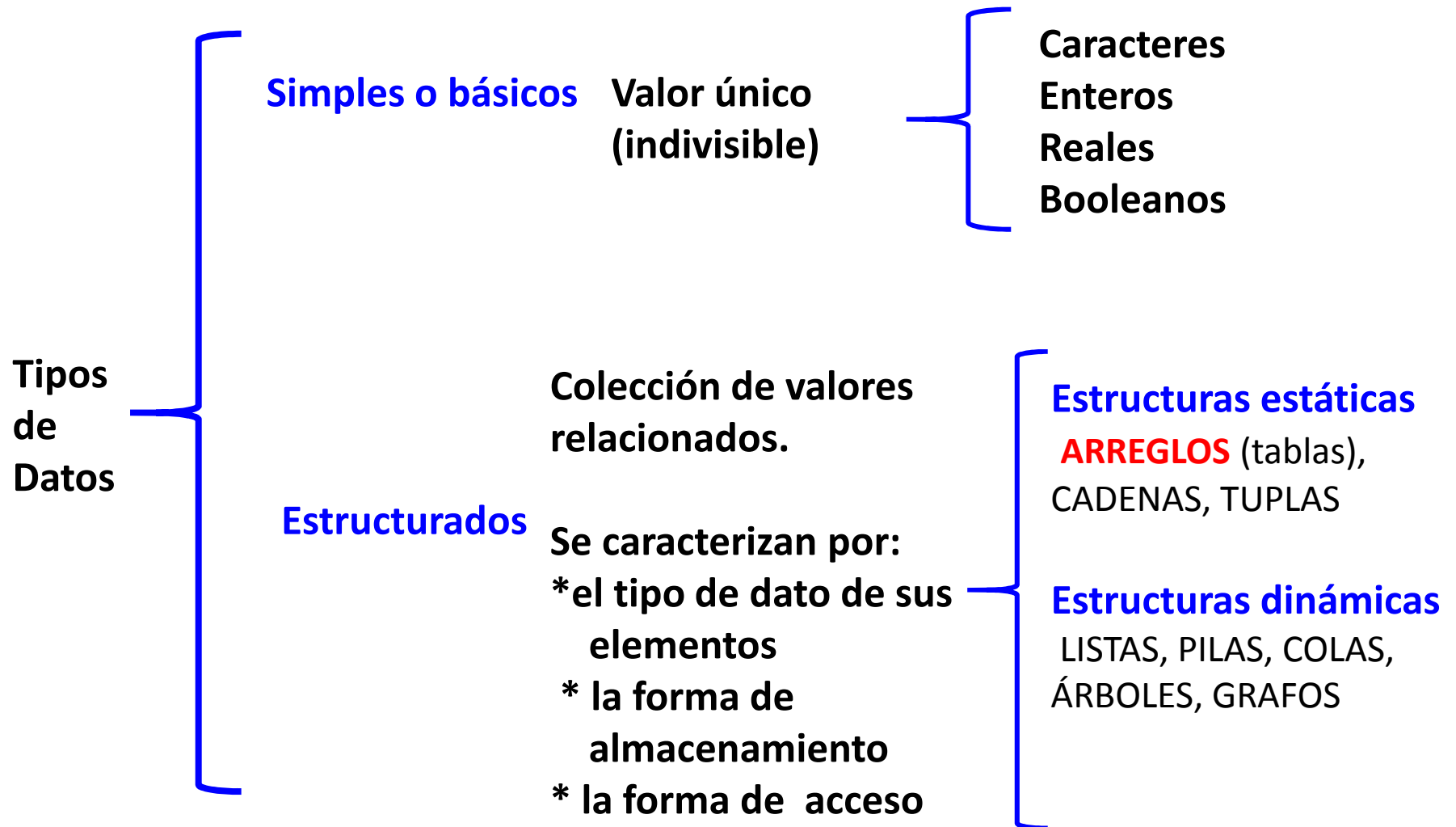
→ La **agregación de elementos del mismo tipo**, que se conocen como “**Agregados**”, “**Arreglos**” o mediante su nombre en inglés “**Arrays**”.

Además de estos tipos compuestos definidos por el programador, los lenguajes de programación suelen proporcionar algún tipo adicional para representar “**Cadenas de caracteres**”.

ESTRUCTURA DE DATOS

ARREGLOS

TIPOS DE DATOS



ESTRUCTURAS DE DATOS: Clasificaciones

- Según donde se almacenan
 - Internas** (en memoria principal)
 - Externas** (en memoria auxiliar)
- Según tipos de datos de sus componentes
 - Homogéneas** (todas del mismo tipo)
 - No homogéneas** (pueden ser de distinto tipo)
- Según la implementación
 - Provistas por los lenguajes** (básicas)
 - Abstractas** (TDA - Tipo de dato abstracto que puede implementarse de diferentes formas)
- Según la forma de almacenamiento
 - Estáticas** (ocupan posiciones fijas y su tamaño nunca varía durante todo el módulo)
 - Dinámicas** (su tamaño varía durante el módulo y sus posiciones también)

Veamos algunos problemas:

A- Leer N números e informar si hay más pares o impares

Ej: 1422,345,1545, 1286, 1722, 422, 567, 312 → hay más pares

B- Leer N números e informar cuál es la terminación de una cifra que más se presenta

La cifra de terminación que más aparece es 2

C- Leer N números e informar cuál es la terminación de dos cifras que más se presenta

Las 2 cifras de terminación que más aparece es 22

D- Y si fueran de 3 cifras???? Es 422


Veamos qué variables se requieren para solucionarlos:

A- Leer N números e informar si hay más pares o impares

 cpar

 cimpar

B- Leer N números e informar cuál es la terminación de una cifra que más se presenta

 c0

 c1

.....

 c9

C- Leer N números e informar cuál es la terminación de dos cifras que más se presenta

 c0

 c1

.....

 c99

D- Y si fueran de 3 cifras????

Estructura de Datos: ARREGLO

- Secuencia en memoria de un número determinado de datos del mismo tipo (**tipo base**: cualquier tipo simple o estructuras definidas por el usuario).
- Los datos se llaman **elementos** (**celdas o componentes**) del arreglo y se numeran consecutivamente 0,1,2,3,4, etc. Esos números se denominan **valores índice** o **subíndice** del arreglo y localizan la posición del elemento dentro del arreglo.
- Los elementos del array se almacenan siempre en **posiciones consecutivas** de la memoria.
- Los elementos se **acceden en forma directa**, indicando el nombre del array y el subíndice (**variables indizadas**).
- Pueden ser **unidimensionales** o **multidimensionales** (accederse con un solo índice o más).

Arreglos

Nombre del arreglo (Todos los elementos del arreglo tienen el mismo nombre, c).

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

- Los índices de arreglos en C++ siempre tienen límite inferior 0 (**indexación basada en cero**), y como límite superior el tamaño del arreglo menos 1.

- Para referirse a un elemento, hay que especificar:

- **Nombre del arreglo**
- **Número de la posición**

El primer elemento está en la posición 0: c[0]

El segundo elemento está en la posición 1: c[1]

El quinto elemento está en la posición 4: c[4]

El elemento n está en la posición n-1: c[n-1]

Número de posición del elemento dentro del arreglo c.

Arreglos: Declaración

- Los arreglos se deben declarar antes de utilizarse.
- Formato de declaración:

tipo nombre-del-arreglo [tamaño]

Por ejemplo, para crear un array de nombre *c*, de doce variables enteras, se lo declara de la siguiente manera:

```
int c [12];
```

El array *c* contiene 12 elementos: el primero es *c[0]* y el último es *c[11]*.

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Arreglos: Declaración

- Las **operaciones posibles de realizar con cada elemento del array** son todas las posibles de realizar con datos del **tipo base** del arreglo.
- El tamaño de un arreglo puede ser un valor constante o representado por una constante:

```
#define N 12  
int c [N];
```

- Ejemplos de declaraciones de arrays:**


```
float  ingresosMensuales [12];      // Declara un array de 12 elementos float.  
int    notasParcial [60], salariosEmple [1200];    // Declara 2 arrays de enteros.  
char   inicialNombre [45];         // Declara un array de caracteres.  
float  alturaRio [365];            // Declara un array de 365 elem. float.  
char   nombre[15];                // Declara un array de caracteres
```

- Se puede definir un **tipo tabla unidimensional**:

```
typedef float tNotas [30];  
.....  
tNotas curso;
```

Arreglos: Almacenamiento

■ Un arreglo tiene tres partes:

- 
- **La dirección** (ubicación) en memoria de la primera variable indizada
 - **El tipo base del arreglo** (que determina cuánta memoria ocupa cada variable indizada)
 - **El tamaño del arreglo** (cantidad de elementos del arreglo)

■ Importante: C++ **no comprueba** que los índices estén dentro del rango definido (posible fallo).

Cuestiones de tamaño

- El **tamaño físico de los arreglos siempre se determina** en tiempo de compilación (no de ejecución).
 - En un arreglo se puede almacenar una **cantidad de elementos menor o igual que la declarada en su capacidad**, nunca mayor.
 - Si se almacenan menos elementos de los que caben se necesita alguna **variable auxiliar** que permita saber, en todo momento, cuántas de las celdas contienen información válida. (**tamaño real o actual**)
-

- **Tamaño de los arrays:** El operador **sizeof** devuelve el número de bytes reservados para el array completo.

Ej: para `int c [12];` `aux = sizeof (c);` // Devolverá el valor 48

- Para conocer el **tamaño de un elemento individual del array**:
Ej: `aux = sizeof (c[3]);` // Retorna 4 (número de bytes de un entero)

Arreglos: Referencia e inicialización

- **Referenciar a los elementos** del arreglo: Se los puede referenciar por medio del subíndice, o utilizando expresiones con resultado entero:

`c[3]` `c[j]` `c[j+3]` `c[m[i] +1]`

- **Inicialización de arrays**: Antes de utilizarse, los arrays deben inicializarse:

- **En la declaración:**

```
int a [3] = {12,34,45};
```

```
int b [ ] = {4,6,7};
```

```
char c[ ] = {'l', 'a', 's', 'f'};
```

```
/* Declara un array de 3 elementos */
```

```
/* Declara un array de 3 elementos */
```

```
/* Declara un array de 4 elementos */
```

```
/* Es el método más utilizado. Inicializa  
todos los valores del array a al valor 0 */
```

- **Con asignaciones:**

```
for (i=0; i<3; i++)
```

```
    a[i]=0;
```

- **Desde la entrada:**

```
for (i=0; i<3; i++)
```

```
    cin >> a[i];
```

```
/* Inicializa los valores del array a utilizando  
los valores ingresados por el usuario */
```

Arreglos: Referencia e inicialización

- Inicialización de variables de tipo array :

```
typedef float tNuevo[3];
```

```
.....
```

```
tNuevo a = {12, 34, 45};
```

```
tNuevo b = {4, 6, 7};
```

Inicializar- Cargar un arreglo

int A [6]

- Puede cargarse en forma completa al declarar la variable (inicializar):

int A[6] = {1,3,5,7,9,11}

1	3	5	7	9	11
0	1	2	3	4	5

- Puede cargarse **elemento a elemento**:

A[0]=1; A[1]=3; A[2]=5; A[3]=7; A[4]=9; A[5]=11;

**Estas sentencias pueden ser consecutivas
o estar entre otras sentencias,
y pueden estar en cualquier orden**

- Si hay una recurrencia, puede cargarse con un ciclo:

**for (j=0; j<6;j++)
A[j] = 2*j +1;**

Inicializar- Cargar un arreglo

`int A [6]`

- Puede leerse desde entrada, en el mismo orden que se van a almacenar:

```
for (j=0; j<6;j++)  
    cin >> A[j] ;
```

6	1	4	4	0	1
0	1	2	3	4	5

Ingreso : 6 1 4 4 0 1

- Puede **leerse desde la entrada**, pero no necesariamente en su orden. Para ello se debe ingresar también la posición:

```
for (j=0; j<6;j++) {  
    cin >> k >> A[k] ;  
    cout << endl;  
}
```

Ingreso :

5 1
1 1
0 6
2 4
3 4
4 0

Inicializar-Cargar un arreglo

int A [6]

- Puede **cargarse con información que se va produciendo durante el proceso**:

Si el arreglo contendrá la cantidad de veces que aparecen los números de 0 a 5 en una secuencia de entrada, la carga se realiza:

```
for (j=0; j<6;j++) A[j]=0;
```

.....

```
for (i=0; i<N; i++) {  
    cin >> k;  
    if (k>=0) && (k<=5) A[k]++ ;  
}
```

Ingreso: 1 7 2 0 5 2 8 2

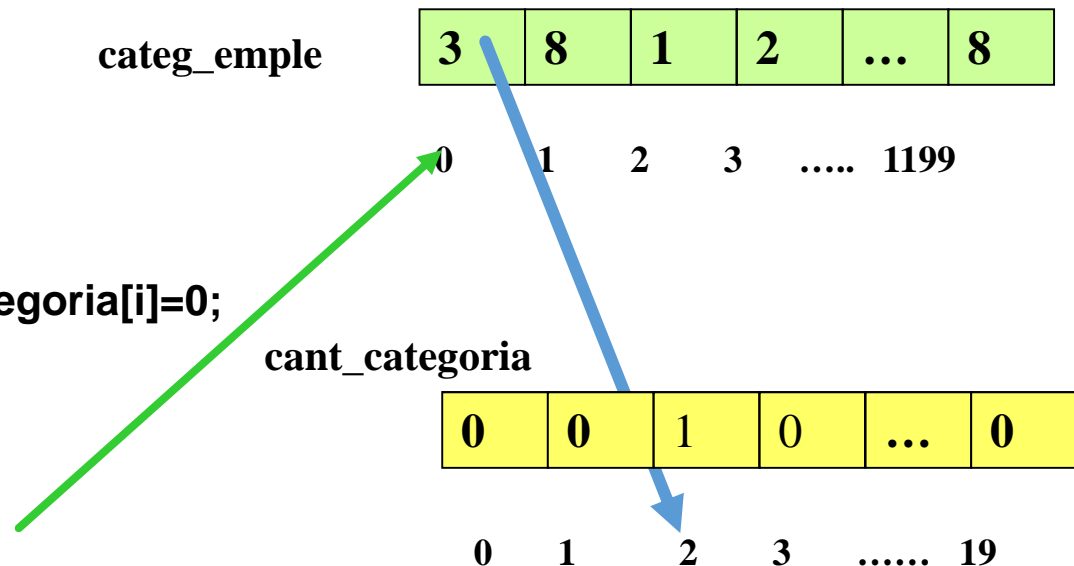
0	0	0	0	0	0
0	1	2	3	4	5

1	1	3	0	0	1
0	1	2	3	4	5

Ejemplo 1. Cargar un arreglo

Tenemos un arreglo donde se almacenan las categorías de 1200 empleados y un arreglo para almacenar la cantidad de empleados de cada categoría (de 1 a 20).

```
int main ( ) {  
    int categ_emple [1200];  
    int cant_categoria [20];  
    int i;  
    .....  
    for (i=0; i<20; i++) cant_categoria[i]=0;  
  
    for (i=0; i<1200; i++)  
        cin >> categ_emple[i];  
  
    for (i=0; i<1200; i++){  
        cant_categoria [categ_emple [i]-1 ] ++;  
    }  
}
```



Arreglos: Almacenamiento

- Declaración: **int c [12]**: hace que el compilador reserve espacio suficiente para contener 12 valores enteros.
- Los elementos de los arrays se almacenan en bloques contiguos.
- Espacio ocupado por c: 48 bytes, pues posee 12 elementos de tamaño $t = 4$ bytes.
- Direcciones:

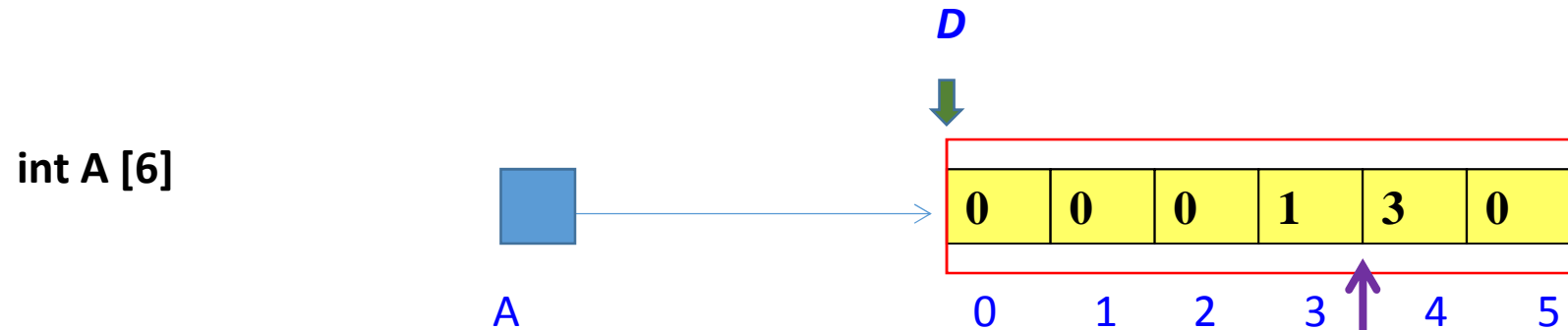
$$\&c[0] = 1000 = D$$

$\&c[i] = D + i * t$

 $\rightarrow \&c[4] = 1000 + 4 * 4 = 1016$

c[0]	-45	1000
c[1]	6	1004
c[2]	0	1008
c[3]	72	1012
c[4]	1543	1016
c[5]	-89	1020
c[6]	0	1024
c[7]	62	1028
c[8]	-3	1032
c[9]	1	1036
c[10]	6453	1040
c[11]	78	1044

Arreglos en C++ - Acceso directo



La variable `A` refiere a un espacio de la memoria donde están almacenados en forma contigua los elementos del arreglo.

El contenido de `A` es la dirección donde comienza a almacenarse el arreglo, que también es la dirección del primer elemento del arreglo

$$\&A = \&A[0] = D$$

El acceso a cualquier elemento del arreglo tiene la misma complejidad (requiere el mismo tiempo) ya que cuando se lo referencia, se hace un cálculo y se lo extrae de la dirección correspondiente:

$$\&A[i] = D + i * t \quad \rightarrow \quad \&A[4] = D + 4 * 4 = D + 16$$

Ejemplo 2. Puntajes

```
/* Lee 5 puntajes e indica la diferencia entre
cada puntaje y el puntaje más alto. */
#include <iostream>
using namespace std;

const int NUM_ESTUDIANTES = 5;

int main() {
    int i, puntos[NUM_ESTUDIANTES], max;
    cout << "Escriba 5 puntajes: " << endl;
    cin >> puntos[0];
    max = puntos[0];
    for (i = 1; i < 5; i++){
        cin >> puntos[i];
        if (puntos[i] > max)
            max = puntos[i];
        //max es el mayor de los valores puntos[0], ..., puntos[i].
    }
    cout << "El puntaje mas alto es " << max << endl
         << "Los puntajes y sus diferencias " << endl
         << "respecto al mas alto son: " << endl;
    for (i = 0; i < 5; i++){
        cout << puntos[i] << ", la diferencia es de "
             << (max - puntos[i]) << endl;
    }
    return 0;
}
```

Escriba 5 puntajes:

```
1
5
3
2
3
El puntaje mas alto es 5
Los puntajes y sus diferencias
respecto al mas alto son:
1, la diferencia es de 4
5, la diferencia es de 0
3, la diferencia es de 2
2, la diferencia es de 3
3, la diferencia es de 2
```

<< El programa ha finalizado: código

Ejemplo de puntajes – agregados

- A- Suponiendo que los competidores están numerados de 1 a 5, permitir que se ingresen los puntajes renglón a renglón como dupla: (competidor- puntaje) en cualquier orden.
- B- Mostrar el número del competidor que ganó
- C- Indicar si hubo empate
- D- Mostrar el puntaje promedio
- E- Informar los números de los competidores que estuvieron por debajo del promedio
- F- Permitir que puedan cargarse 100 competidores
- G- Permitir que puedan cargarse hasta 100 competidores.
- H- Se tiene cargados los competidores que tendrán un descuento del 15% de su puntajes. Se deben consultar para actualizar el puntaje final.
- I- Mostrar la tabla completa de puntajes ordenada.

Arreglos Lineales: Operaciones

Las siguientes son las operaciones más comunes en programas que manejan arreglos:

- > Imprimir los elementos de un arreglo
- > Hallar el promedio de los datos de un arreglo
- > Insertar un elemento en un arreglo
- > Borrar un elemento de un arreglo (conociendo el índice o el valor)
- > Borrar un rango
- > Buscar un elemento en un arreglo.

Operación: Imprimir elementos

■ Ejemplo 3. Imprimir los elementos de un arreglo

Con un ciclo exacto desde 0 hasta la última posición del arreglo, mostrar el índice y su contenido.

Se carga un arreglo con los 5 primeros naturales pares

Se muestran los títulos

Se imprimen los valores con sus respectivos índices

```
#include <iostream>
using namespace std;
# define Tam 5

int main(){
    int c[Tam];

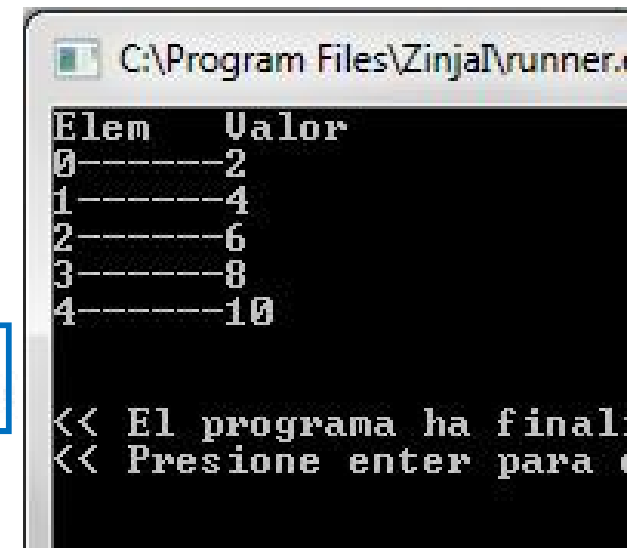
    int i;

    for (i=0; i<Tam; i++)
        c[i] = 2 + 2*i;

    cout << "Elem" << " Valor" << endl;

    for (i=0; i<Tam; i++)
        cout << i << "-----" << c[i] << endl;

    return 0;
}
```



```
C:\Program Files\Zinjal\runner...
Elem  Valor
0-----2
1-----4
2-----6
3-----8
4-----10

<< El programa ha finalizado
<< Presione enter para continuar
```


Arreglos Lineales: Tamaño físico y tamaño real

- Puede ocurrir que al compilar, se conozca el **tamaño máximo** que puede tener un arreglo, pero no el **tamaño real**, el que se conocerá en tiempo de ejecución y que incluso puede variar de un momento a otro.
- Es necesario entonces definir una variable que contenga en todo momento el tamaño actual del mismo (subíndice del último elemento cargado).
- Se debe tener en cuenta que los recorridos del arreglo deben entonces realizarse teniendo como tope a dicho límite.

Operación: Hallar el promedio de los elementos

■ Ejemplo 4. Hallar el promedio de los datos de un arreglo

```
#include <iostream>
using namespace std;
# define Tam 10

int main(){
    int c[Tam];

    int Fin,i,s=0;

    cout << "Ingrese el tamaño: ";
    cin >> Fin;

    if (Fin > 0 && Fin < Tam){
        for (i=0; i<Fin; i++){
            cout << endl << "Ingrese el elemento " << i << ": ";
            cin >> c[i];
        }

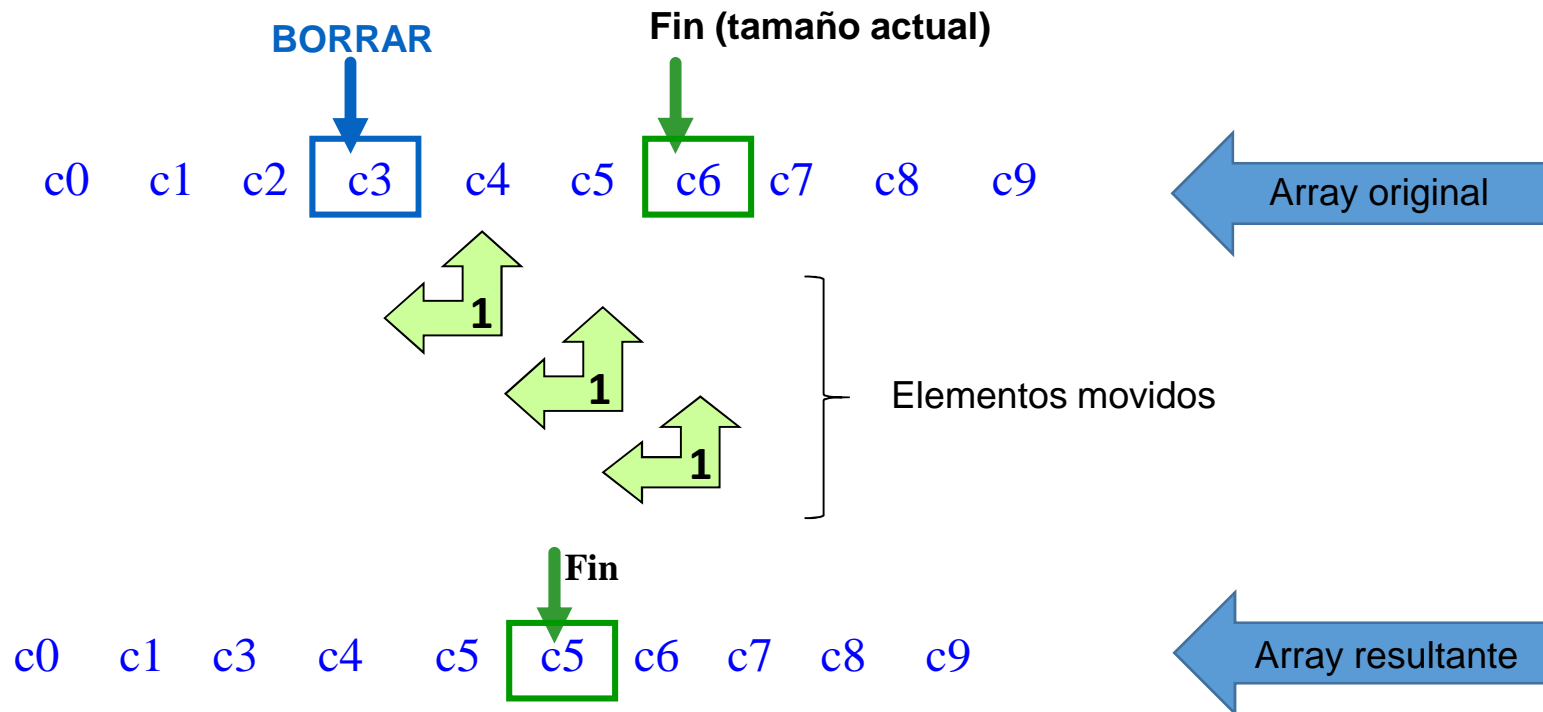
        for (i=0; i<Fin; i++)
            s = s + c[i];
        cout << endl << "Promedio: " << (float)s/Fin;

    }
    return 0;
}
```

```
Ingrese el tamaño: 4
Ingrese el elemento 0: 3
Ingrese el elemento 1: 5
Ingrese el elemento 2: 6
Ingrese el elemento 3: 7
Promedio: 5.25
<< El programa ha finalizado: co
<< Presione enter para cerrar es
```

Operación: Borrar un elemento de un arreglo

- **Borrar un elemento de un arreglo:** implica correr los elementos que están a la derecha una posición hacia la izquierda, comenzando por el más cercano del elemento a borrar.
- Ejemplo: Tamaño físico = 10; Tamaño actual = 7; Borrar el 4to elemento



Operación: Borrar un elemento de un arreglo conociendo su posición (Ejemplo 5)

```
#include <iostream>
using namespace std;
#define Tam 10

int main(){
    int c[Tam];

    int Fin,i,b;

    cout << "Ingrese el tamaño: ";
    cin >> Fin;

    if (Fin > 0 && Fin < Tam){
        for (i=0; i<Fin; i++){
            cout << endl << "Ingrese el elemento " << i << ": ";
            cin >> c[i];
        }

        cout << endl << "Ingrese la posición del elemento a borrar: ";
        cin >> b;
        for (i=b; i<Fin-1; i++){
            c[i] = c[i+1];
        }
        Fin--;
        cout << endl << "Luego de borrar: " << endl;
        for (i=0; i<Fin; i++)
            cout << "C[" << i << "] = " << c[i] << "; ";

    }

    return 0;
}
```

```
Ingrese el tamaño: 5
Ingrese el elemento 0: 1
Ingrese el elemento 1: 2
Ingrese el elemento 2: 3
Ingrese el elemento 3: 4
Ingrese el elemento 4: 5
Ingrese la posición del elemento a borrar: 2

Luego de borrar:
C[0] = 1; C[1] = 2; C[2] = 4; C[3] = 5;

<< El programa ha finalizado: código de salida:
<< Presione enter para cerrar esta ventana >>
```

Borrar un
elemento
conociendo su
posición

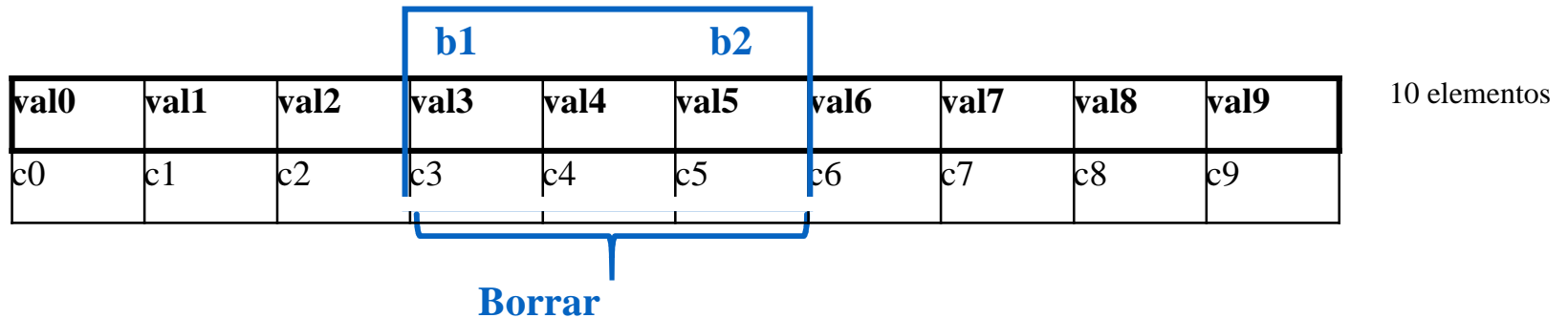
Operación: Borrar un elemento de un arreglo conociendo su posición

```
cout << endl << "Ingrese la posicion del elemento a borrar: ";  
cin >> pos;  
for (i=pos; i<Fin-1; i++)  
    c[i] = c[i+1];  
Fin--;
```

**Borrar un
elemento
conociendo su
posición**

Operación: Borrar un rango

- **Borrar un rango:** A diferencia de borrar un solo elemento de subíndice b debemos borrar desde $b1$ hasta $b2$. Por lo tanto, los elementos siguientes a $b2$, hasta el fin del array, deberán correrse $(b2 - b1 + 1)$ posiciones a la izquierda.



$$b2 - b1 + 1 = c5 - c3 + 1 = 2 + 1 = 3 \text{ posiciones se deben correr}$$

val0	val1	val2	val6	val7	val8	val9			
c0	c1	c2	c3	c4	c5	c6	c7	c8	c9

7 elementos

Operación: Borrar un rango

```
int cant = b2-b1+1;  
  
    for (j=b2+1; j<=tam-1; j++)  
  
        vec[j-cant] = vec[j];  
  
    tam = tam-cant;  
  
    return ;
```

Operación: Buscar un elemento en un arreglo

- El algoritmo difiere si el arreglo está o no ordenado por algún criterio (ascendente o descendente).
- Si no hay orden, deben examinarse los elementos de izquierda a derecha (o de derecha a izquierda) hasta encontrar el elemento buscado o hasta que se hayan examinado todos los elementos

Búsqueda Secuencial.

- Si hay orden, debe examinarse el elemento central del arreglo, si el elemento a buscar es menor que éste, se buscará en la primera mitad del arreglo, sino se buscará en la segunda mitad. Se repite esta acción hasta que se encuentre el elemento o hasta que se obtenga una mitad consistente de un solo elemento que no es el buscado

Búsqueda Binaria.

Operación: Buscar un elemento

Buscar un elemento en un arreglo: Dados un arreglo $A[N]$ y un elemento X , determinar si existe algún i , tal que $0 \leq i < N$ y $A[i] = X$

Si A no está ordenado: **Búsqueda Secuencial:** se examinan los elementos desde el primero, continuando con el segundo, y así sucesivamente hasta encontrar el buscado ó hasta que se hayan examinado todos los elementos:

```
i=0
while (i<N ) and (A[i] != X)
    i++;

if (i>=N) cout << "X no encontrado";
else cout << "X encontrado en posición " << i );
```

Mejor caso: 0 pasadas del ciclo while; **Peor caso:** N pasadas del ciclo while.

Operación: Buscar un elemento

■ Ejemplo 6: Buscar un elemento en un arreglo usando Búsqueda Lineal

```
//Esta funcion devuelve la ubicacion de clave en la lista
//Se devuelve un -1 si no se encuentra el valor
int busquedaLineal(int lista[], int tamano, int clave){
    int i;

    for (i = 0; i < tamano; i++){
        if (lista[i] == clave)
            return i;
    }
    return -1;
}
```

Otra
alternativa
usando for

Ver que el return hace que el ciclo
for se interrumpa

Operación: Buscar un elemento

Buscar un elemento en un arreglo: Dados un arreglo $A[N]$ y un elemento X , determinar si existe algún i , tal que $0 \leq i < N$ y $A[i] = X$

Si A está ordenado: **Búsqueda Binaria:** se examina el elemento central, si no es el buscado, se determina en que mitad (izquierda o derecha) puede estar el elemento buscado y se pasa a examinar esa mitad, repitiendo este criterio hasta encontrar el valor buscado o hasta que el intervalo de búsqueda sea menor que 1.

```
izq=0; der=N-1
med = (izq + der)/2
while (izq <= der ) and (A [med] != X)
    If (X < A[med])
        der=med-1
    else
        izq=med+1
    med = (izq + der)/2

if (der < izq) "X no encontrado"
else "X encontrado en posición med"
```

Mejor caso:

0 pasadas del ciclo while

Peor caso:

$(\log_2 N)$ pasadas del ciclo while

Operación: Buscar un elemento

■ Ejemplo 7: Buscar un elemento en un arreglo usando Búsqueda Binaria

```
// esta función devuelve la ubicación de clave en la lista
// se devuelve -1 si no se encuentra el valor
int busquedaBinaria(int lista[], int tamaño, int clave)
{
    int izquierdo, derecho, puntomedio;

    izquierdo = 0;
    derecho = tamaño - 1;

    while (izquierdo <= derecho)
    {
        puntomedio = (int) ((izquierdo + derecho) / 2);
        if (clave == lista[puntomedio])
        {
            return puntomedio;
        }
        else if (clave > lista[puntomedio])
            izquierdo = puntomedio + 1;
        else
            derecho = puntomedio - 1;
    }

    return -1;
}
```

Operación: Borrar un elemento conociendo el valor a borrar

- **Situación 1:** El arreglo no está ordenado, no hay elementos repetidos y hay que borrar el elemento de valor V.
Estrategia: Recorrido secuencial del arreglo que se detiene cuando se encuentra el elemento y desde allí hasta el final se corre una posición.
- **Situación 2:** El arreglo no está ordenado, puede haber elementos repetidos y hay que borrar el elemento de valor V todas las veces que aparece.
Estrategia: Recorrido secuencial del arreglo que se detiene cuando se encuentra el 1er valor, luego se corren a la izquierda todos los valores a la derecha de V y se repite el proceso hasta eliminar todos los elementos de valor V.
- **Situación 3:** El arreglo está ordenado, no hay repetidos y hay que borrar el elemento de valor V.
Estrategia: Se lo busca con un método más eficiente que en la situación 1 y se lo elimina.
- **Situación 4:** El arreglo está ordenado, puede haber repetidos y hay que borrar el elemento de valor V todas las veces que aparece.
Estrategia: Se lo busca con un método más eficiente, luego se analiza a la izquierda y derecha los repetidos (guardando el menor y mayor índice) y luego se borra el rango.

Arreglos en funciones

Arreglos enteros como argumentos de funciones:

Una función puede tener **un parámetro formal para un arreglo entero**: cuando se invoque la función el argumento que se inserte en lugar de este parámetro formal sea un arreglo entero.

Sin embargo, el parámetro formal de un arreglo entero no es **ni de llamada por valor ni de llamada por referencia**; es un nuevo tipo de parámetro formal llamado **parámetro de arreglo**.

Arreglos en funciones

Ejemplo:

el parámetro formal `int a[]` es un parámetro de arreglo.

Los corchetes sin expresión índice adentro , son lo que C++ usa para indicar un parámetro de arreglo.

Un **parámetro de arreglo** no es exactamente un parámetro de llamada por referencia, pero en la práctica **se comporta de forma muy parecida a uno.**

Cuando usamos un arreglo como argumento en una llamada de función, cualquier acción que se efectúe con el parámetro de arreglo se efectúa con el argumento arreglo, así que la función puede modificar los valores de las variables indizadas del argumento arreglo.

Arreglos en funciones

```
void llenar(int a[], int tamano);
```

```
void llenar(int a[], int tamano)
{
    cout << "Teclee " << tamano << " numeros:\n";
    for (int i = 0; i < tamano; i++)
        cin >> a[i];
    tamano--;
    cout << "El ultimo indice de arreglo empleado fue " << tamano << endl;
}
```

```
int puntos[5], numero_de_puntajes = 5;
llenar(puntos, numero_de_puntajes);
```

Es necesario pasar
como parámetro el
tamaño del arreglo

```
llenar(puntos, 5);
llenar(tiempo, 10);
```

Arreglos de argumentos de
diferente tamaño pueden sustituir
al mismo parámetro de arreglo

■ Ejemplo 9

Para trabajar con una copia, hay que hacerlo explícitamente

```
#define K 8
typedef int vect[K];
void pasarArreglo(vect a, int tamano);

int main(){
    vect x;
    int i, N;
    cout << "Ingrese tamano (hasta 10): ";
    cin >> N;
    cout << endl << "Ingrese el arreglo:" << endl;
    for (i=0; i<N; i++)
        cin >> x[i];
    pasarArreglo(x,N);

    cout << endl << endl << "Arreglo x: " << endl;
    for (i=0; i<N; i++)
        cout << x[i] << " ";

    return 0;
}

void pasarArreglo(vect a, int tamano){
    int i;
    vect z;
    for (i=0; i<tamano; i++)
        z[i] = a[i];

    cout << endl << "Arreglo z: " << endl;
    for (i=0; i<tamano; i++)
        cout << z[i] << " ";
    for (i=0; i<tamano; i++)
        z[i] = 0;
    return;
}
```

```
Ingrese tamano (hasta 10): 5
Ingrese el arreglo:
1 2 3 4 5
Arreglo z:
1 2 3 4 5
Arreglo x:
1 2 3 4 5
<< El programa ha finalizado: con
```

OJO!

Se pasa un parámetro arreglo (a) y se copia (en z) para modificarlo pero con alcance local a la función

Arreglos en funciones

Parámetro de arreglo constante

Cuando usamos un argumento arreglo en una llamada de función, la función puede modificar los valores almacenados en el arreglo.

Pero podemos indicarle a la computadora que no pensamos modificar el argumento arreglo, y entonces la computadora se asegurará de que el código no modifique inadvertidamente algún valor del arreglo.

```
void mostrar(const int a[], int tamano_de_a)
{
    cout << "El arreglo contiene los siguientes valores:\n";
    for (int i = 0; i < tamano_de_a; i++)
        cout << a[i] << " ";
    cout << endl;
}
```

Esta sentencia será detectada como error:

```
for (int i = 0; i < tamano_de_a; a[i]++)
```

Arreglos en funciones

Funciones que devuelven arreglos:

Un arreglo no se puede devolver de la misma forma que un entero o un real .

Hay una manera para obtener algo más o menos equivalente a una función que devuelve un arreglo.

Lo que se debe hacer es devolver un apuntador a un arreglo.

Lo veremos luego.

Borrado en arreglos

**Veamos soluciones para los problemas
planteados**

Operación: Borrar un elemento conociendo el valor a borrar

- **Situación 1:** El arreglo no está ordenado, no hay elementos repetidos y hay que borrar el elemento de valor V.
- Estrategia: Recorrido secuencial del arreglo que se detiene cuando se encuentra el elemento y desde allí hasta el final se corre una posición.
- **Situación 2:** El arreglo no está ordenado, puede haber elementos repetidos y hay que borrar el elemento de valor V todas las veces que aparece.
- Estrategia: Recorrido secuencial del arreglo que se detiene cuando se encuentra el 1er valor, luego se corren a la izquierda todos los valores a la derecha de V y se repite el proceso hasta eliminar todos los elementos de valor V.
- **Situación 3:** El arreglo está ordenado, no hay repetidos y hay que borrar el elemento de valor V.
- Estrategia: Se lo busca con un método más eficiente que en la situación 1 y se lo elimina.
- **Situación 4:** El arreglo está ordenado, puede haber repetidos y hay que borrar el elemento de valor V todas las veces que aparece.
- Estrategia: Se lo busca con un método más eficiente, luego se analiza a la izquierda y derecha los repetidos (guardando el menor y mayor índice) y luego se borra el rango.

- **Situación 1:** El arreglo no está ordenado, no hay elementos repetidos y hay que borrar el elemento de valor V.
- **Estrategia:** Recorrido secuencial del arreglo que se detiene cuando se encuentra el elemento y desde allí hasta el final se corre una posición.

```
cout << "Elemento a buscar? ";
cin >> elem;

pos = busqueda(vector, TL, elem);
if (pos >= 0 ) {
    borrar(vector, TL, pos);
}
else
    cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
```

- **Situación 1:** El arreglo no está ordenado, no hay elementos repetidos y hay que borrar el elemento de valor V.
- **Estrategia:** Recorrido secuencial del arreglo que se detiene cuando se encuentra el elemento y desde allí hasta el final se corre una posición.

```
cout << "Elemento a buscar? ";
cin >> elem;

pos = busqueda(vector, TL, elem);
if (pos>=0)
    borrar(vector, TL, pos);
else
    cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
```

```
int busqueda(const int vec [], const int tl, const int valor){
    int esta=0;
    int i=0;
    int pos=-1;

    while (!esta && i<tl){
        if (vec[i] == valor) {
            esta = 1;
            pos = i;
        }
        else
            i++;
    }
    return pos;
}
```

```
void borrar(int vec [], int & tl, int pos){
    int i;
    for (i=pos; i<tl-1; i++)
        vec[i] = vec[i+1];
    tl--;
    return ;
}
```

Situación 1

Programa de Prueba

```
const int TF=20;

void cargarvector(int vec [], const int tl);
void mostrarvector(const int vec [], const int tl);
int busqueda(const int vec [], const int tl, const int valor);
void borrar(int vec [], int & tl, int pos);

int main(){
    int vector[TF];
    int TL, elem, pos;

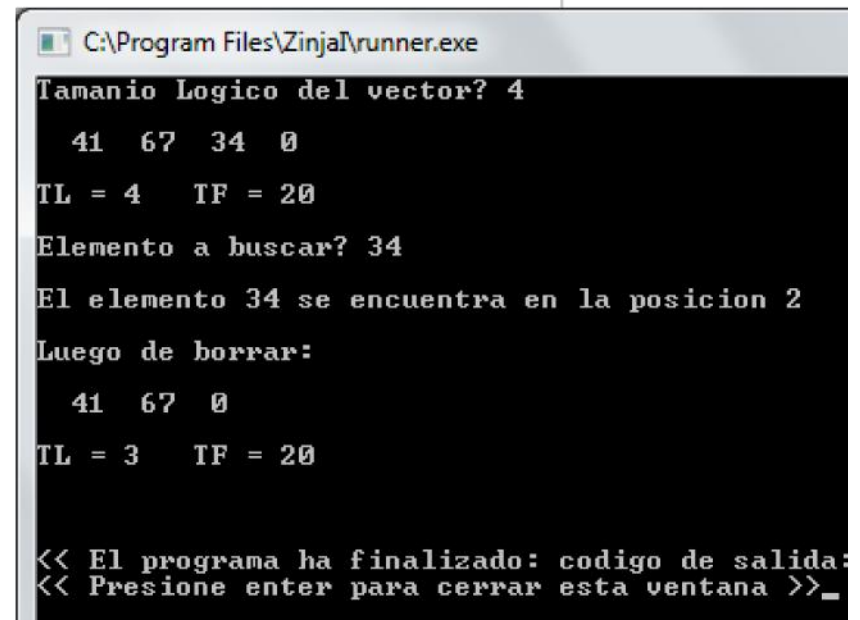
    cout << "Tamano Logico del vector? ";
    cin >> TL;

    cargarvector(vector, TL);
    mostrarvector(vector, TL);

    cout << "Elemento a buscar? ";
    cin >> elem;

    pos = busqueda(vector, TL, elem);
    if (pos>=0){
        cout << endl << "El elemento " << elem << " se encuentra en la ";
        cout << "posicion " << pos;
        borrar(vector, TL, pos);
        cout << endl << endl << "Luego de borrar: " << endl;
        mostrarvector(vector, TL);
    }
    else
        cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";

    return 0;
}
```

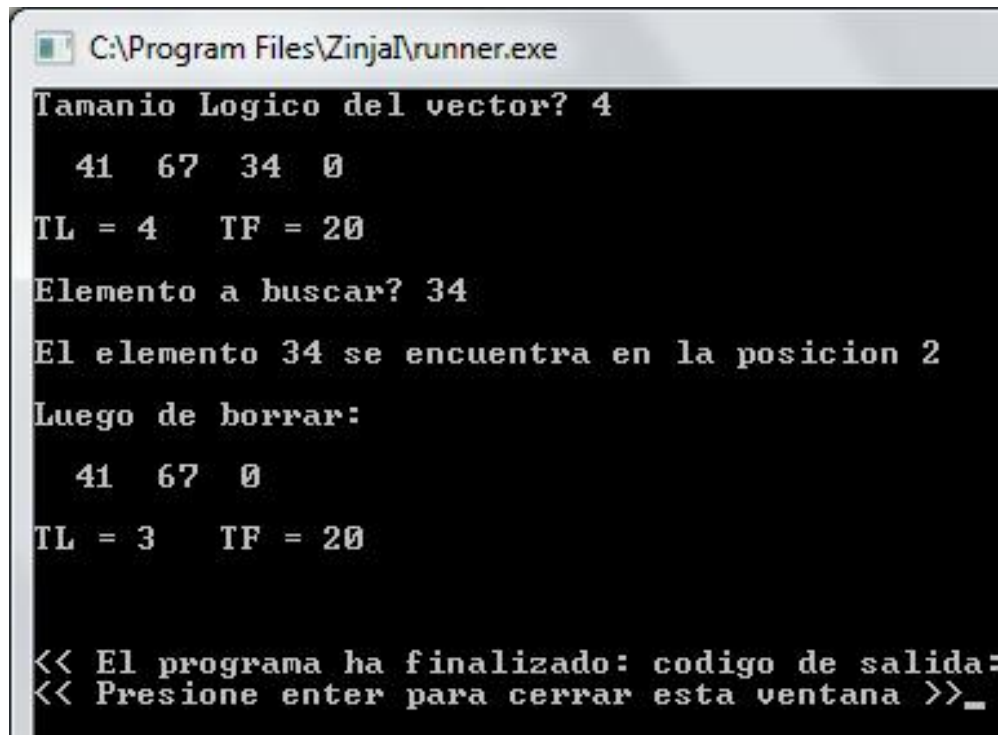


```
C:\Program Files\Zinja\runner.exe
Tamano Logico del vector? 4
41 67 34 0
TL = 4 TF = 20
Elemento a buscar? 34
El elemento 34 se encuentra en la posicion 2
Luego de borrar:
41 67 0
TL = 3 TF = 20
<< El programa ha finalizado: codigo de salida:
<< Presione enter para cerrar esta ventana >>_
```


Situación 1
Programa de
Prueba

La carga del vector es aleatoria

```
void cargarvector(int vec[], const int tl){  
    int i;  
    for (i=0; i<tl; i++)  
        vec[i] = rand()%100;  
}
```



```
C:\Program Files\Zinja\runner.exe  
Tamano Logico del vector? 4  
    41  67  34  0  
TL = 4    TF = 20  
Elemento a buscar? 34  
El elemento 34 se encuentra en la posicion 2  
Luego de borrar:  
    41  67  0  
TL = 3    TF = 20  
  
<< El programa ha finalizado: codigo de salida:  
<< Presione enter para cerrar esta ventana >>_
```

- **Situación 2:** El arreglo no está ordenado, puede haber elementos repetidos y hay que borrar el elemento de valor V todas las veces que aparece.
- **Estrategia:** Recorrido secuencial del arreglo que se detiene cuando se encuentra el 1er valor, luego se corren a la izquierda todos los valores a la derecha de V y se repite el proceso hasta eliminar todos los elementos de valor V.

```
cout << "Elemento a buscar? ";
cin >> elem;

pos = busqueda(vector, TL, elem);
if (pos < 0)
    cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
else
    while (pos >= 0) {
        borrar(vector, TL, pos);
        pos = busqueda(vector, TL, elem);
    }
```

Situación 2

Programa de Prueba

```
const int TF=20;
void cargarvector(int vec [], const int tl, const int tf);
void mostrarvector(const int vec [], const int tl);
int busqueda(const int vec [], const int tl, const int valor);
void borrar(int vec [], int& tl, int pos);

int main() {
    int vector[TF];
    int TL, elem, pos;
    cout << "Tamano Logico del vector? ";
    cin >> TL;

    cargarvector(vector, TL, TF);
    mostrarvector(vector, TL);

    cout << "Elemento a buscar? ";
    cin >> elem;

    pos = busqueda(vector, TL, elem);
    if (pos < 0)
        cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
    else
        while (pos >= 0) {
            cout << endl << "El elemento " << elem << " se encuentra en la ";
            cout << "posicion " << pos;
            borrar(vector, TL, pos);
            cout << endl << endl << "Luego de borrar: " << endl;
            mostrarvector(vector, TL);
            pos = busqueda(vector, TL, elem);
        }
    return 0;
}
```

```

C:\Program Files\Zinja\runner.exe
Tamano Logico del vector? 4
Ingrese el elemento 0: 1
Ingrese el elemento 1: 2
Ingrese el elemento 2: 1
Ingrese el elemento 3: 1
    1  2  1  1
TL = 4    TF = 20
Elemento a buscar? 1
El elemento 1 se encuentra en la posicion 0
Luego de borrar:
    2  1  1
TL = 3    TF = 20

El elemento 1 se encuentra en la posicion 1
Luego de borrar:
    2  1
TL = 2    TF = 20

El elemento 1 se encuentra en la posicion 1
Luego de borrar:
    2
TL = 1    TF = 20

<< El programa ha finalizado

```

AEDD-UTN-FRSF

Situación 2

Programa de Prueba

La carga del vector
es desde teclado

```

void cargarvector(int vec[], const int tl, const int tf){
    int i;
    if (tl > 0 && tl < tf){
        for (i=0; i<tl; i++){
            cout << endl << "Ingrese el elemento " << i << ": ";
            cin >> vec[i];
        }
    }
}

```

- **Situación 3:** El arreglo está ordenado, no hay repetidos y hay que borrar el elemento de valor V.
- **Estrategia:** Se lo busca con un método más eficiente que en la situación 1 y se lo elimina.

```
pos = busquedabinaria(vector, TL, elem);  
if (pos >= 0){  
    borrar(vector, TL, pos);  
}  
else  
    cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
```

```
int busquedabinaria(const int vec [], const int tam, const int valor){  
    // busqueda binaria en vector ordenado  
    int central, inicio, fin;  
    inicio=0;  
    fin=tam-1;  
  
    while (inicio <= fin){  
        central = (inicio + fin) / 2;  
  
        if (vec[central] == valor)  
            return central;  
        else  
            if (vec[central] > valor)  
                fin = central-1;  
            else  
                inicio = central+1;  
    }  
    return -1;  
}
```


Situación 3

Programa de Prueba

```
const int TF=20;
void cargarvector(int vec [], const int tl);
void mostrarvector(const int vec [], const int tl);
int busquedabinaria(const int vec [], const int tl, const int valor);
void borrar(int vec [], int & tl, int pos);

int main(){
    int vector[TF];
    int TL, elem, pos;

    cout << "Tamaño Logico del vector? ";
    cin >> TL;

    cargarvector(vector, TL);
    mostrarvector(vector, TL);
    cout << "Elemento a buscar? ";
    cin >> elem;

    pos = busquedabinaria(vector, TL, elem);
    if (pos>=0){
        cout << endl << "El elemento " << elem << " se encuentra en la ";
        cout << "posicion " << pos;
        borrar(vector, TL, pos);
        cout << endl << endl << "Luego de borrar: " << endl;
        mostrarvector(vector, TL);
    }
    else
        cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";

    return 0;
}
```

Situación 3

Programa de Prueba

La carga del vector
es por recurrencia

```
C:\Program Files (x86)\Zinjal\runner.exe
Tamano Logico del vector? 6
  0  10  20  30  40  50
TL = 6   TF = 20
Elemento a buscar? 25
El elemento 25 no se encuentra en el arreglo
<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>
```

```
void cargarvector(int vec[], const int tl)
{
    int i;
    vec[0]=0;
    for (i=0; i<tl; i++)
        vec[i] = vec[i-1] + 10;
}
```

```
C:\Program Files (x86)\Zinjal\runner.exe
Tamano Logico del vector? 5
  0  10  20  30  40
TL = 5   TF = 20
Elemento a buscar? 20
El elemento 20 se encuentra en la posicion 2
Luego de borrar:
  0  10  30  40
TL = 4   TF = 20
<< El programa ha finalizado: codigo de salida: 0 >>
```

- **Situación 4:** El arreglo está ordenado, puede haber repetidos y hay que borrar el elemento de valor V todas las veces que aparece.
- **Estrategia:** Se lo busca con un método más eficiente, luego se lo borra, y se repite el proceso hasta que no se encuentren más elementos a ser borrados.

- **ALTERNATIVA A**

```
cout << "Elemento a buscar? ";
cin >> elem;

pos = busquedabinaria(vector, TL, elem);
if (pos < 0)
    cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
else
    while (pos >= 0) {
        borrar(vector, TL, pos);
        pos = busqueda(vector, TL, elem);
    }
}
```


Situación 4-A
Programa de
Prueba

```
const int TF=20;
void cargarvector(int vec [], const int tl, const int tf);
void mostrarvector(const int vec [], const int tl);
void borrar(int vec [], int & tl, int pos);
int busquedabinaria(const int vec [], const int tl, const int valor);
int main() {
    int vector[TF];
    int TL, elem, pos;

    cout << "Tamano Logico del vector? ";
    cin >> TL;

    cargarvector(vector, TL, TF);
    mostrarvector(vector, TL);

    cout << "Elemento a buscar? ";
    cin >> elem;

    pos = busquedabinaria (vector, TL, elem);
    if (pos <0)
        cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
    else
        while (pos>=0){
            borrar(vector, TL, pos);
            cout << endl << endl << "Luego de borrar: " << endl;
            mostrarvector(vector, TL);
            pos = busquedabinaria(vector, TL, elem);
        }

    return 0;
}
```

```
C:\Program Files (x86)\Zinja\runner.exe
Tamano Logico del vector? 6
Ingrese el elemento 0: 1
Ingrese el elemento 1: 2
Ingrese el elemento 2: 2
Ingrese el elemento 3: 2
Ingrese el elemento 4: 5
Ingrese el elemento 5: 6

1 2 2 2 5 6
TL = 6    TF = 20
Elemento a buscar? 2
```

```
Luego de borrar:

1 2 2 5 6
TL = 5    TF = 20

Luego de borrar:

1 2 5 6
TL = 4    TF = 20

Luego de borrar:

1 5 6
TL = 3    TF = 20

<< El programa ha finalizado:
<< Presione enter para cerrar
```

- **Situación 4:** El arreglo está ordenado, puede haber repetidos y hay que borrar el elemento de valor V todas las veces que aparece.
- **Estrategia:** Se lo busca con un método más eficiente, luego se analiza a la izquierda y derecha los repetidos (guardando el menor y mayor índice) y luego se borra el rango.
- **ALTERNATIVA B**

```
pos = busquedabinaria (vector, TL, elem);
if (pos < 0)
    cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
else
    borrarrango (vector, pos, TL);
```

Busca el primero
repetido a la izquierda

Busca el último
repetido a la derecha

Borra el rango de
repetidos

```
void borrarrango(int vec [], int p, int& tam){
    int i, j, d;

    i= p;
    do
        i= i-1;
    while((i>=0) && (vec[i] == vec[p]));
    i=i+1;

    d= p;
    do
        d= d+1;
    while((d<=tam-1) && (vec[d] == vec[p]));
    d=d-1;

    int cant = d-i+1;

    for (j=d+1; j<tam; j++)
        vec[j-cant] = vec[j];
    tam = tam-cant;

    return ;
}
```

- **Situación 4:** El arreglo está ordenado, puede haber repetidos y hay que borrar el elemento de valor V todas las veces que aparece.

Busca el primero
repetido a la izquierda

Busca el último
repetido a la derecha

Borra el rango de
repetidos

```
void borrar_rango(int vec [], int p, int& tam){
    int i, j, d;

    i = p;
    do
        i = i - 1;
    while((i >= 0) && (vec[i] == vec[p]));
    i = i + 1;

    d = p;
    do
        d = d + 1;
    while((d <= tam - 1) && (vec[d] == vec[p]));
    d = d - 1;

    int cant = d - i + 1;

    for (j = d + 1; j < tam; j++)
        vec[j - cant] = vec[j];
    tam = tam - cant;

    return ;
}
```

```

const int TF=20;
void cargarvector(int vec [], const int tl, const int tf);
void mostrarvector(const int vec [], const int tl);
int busquedabinaria(const int vec [], const int tl, const int valor);
void borrarango(int vec [], int p, int & tam);
int main() {
    int vector[TF];
    int TL, elem, pos;

    cout << "Tamano Logico del vector? ";
    cin >> TL;

    cargarvector(vector, TL, TF);
    mostrarvector(vector, TL);

    cout << "Elemento a buscar? ";
    cin >> elem;

    pos = busquedabinaria (vector, TL, elem);
    if (pos <0)
        cout << endl << "El elemento " << elem << " no se encuentra en el arreglo";
    else {
        borrarango (vector, pos, TL);
        cout << endl << endl << "Luego de borrar: " << endl;
        mostrarvector(vector, TL);
    }

    return 0;
}

```

Situación 4-B
Programa de
Prueba


```
C:\Program Files (x86)\Zinja\runner.exe
Tamano Logico del vector? 8

Ingrese el elemento 0: 1
Ingrese el elemento 1: 2
Ingrese el elemento 2: 3
Ingrese el elemento 3: 3
Ingrese el elemento 4: 3
Ingrese el elemento 5: 5
Ingrese el elemento 6: 6
Ingrese el elemento 7: 7

1 2 3 3 3 5 6 7

TL = 8    TF = 20

Elemento a buscar? 3
```

```
Elemento a buscar? 3

Luego de borrar:

1 2 5 6 7

TL = 5    TF = 20

<< El programa ha finalizado:
<< Presione enter para cerrar
```

Tamaño Logico del vector? 6

Ingrese el elemento 0: 1

Ingrese el elemento 1: 1

Ingrese el elemento 2: 1

Ingrese el elemento 3: 4

Ingrese el elemento 4: 5

Ingrese el elemento 5: 6

1 1 1 4 5 6

TL = 6 TF = 20

Elemento a buscar? 1

Luego de borrar:

4 5 6

TL = 3 TF = 20

Tamaño Logico del vector? 5

Ingrese el elemento 0: 1

Ingrese el elemento 1: 2

Ingrese el elemento 2: 3

Ingrese el elemento 3: 4

Ingrese el elemento 4: 5

1 2 3 4 5

TL = 5 TF = 20

Elemento a buscar? 2

Luego de borrar:

1 3 4 5

TL = 4 TF = 20

<< El programa ha finalizado;

Búsqueda de repetidos en arreglos

Encontrar los elementos que se repiten en un arreglo, informarlos y decir cuantas veces se repiten

Alternativas:

- **Arreglo ordenado**

Se va recorriendo y cuando un valor se repite, saltar todos los iguales hasta llegar a otro distinto

EJ: 345555578999999

- **Arreglo sin orden previo**

Para cada elemento ver si se repite pero.....veamos

Ejemplo 12. Repetidos en arreglos ordenados

Encontrar los elementos que se repiten en un arreglo, informarlos y decir cuantas veces se repiten.

```
int main() {
    int i, j, cont;

    // Trabajando con arreglos ordenados
    int m[15]={3,3,3,6,6,6,7,8,8,9,10,16,16,30,32};
    //
    // ----->
    // !
    // ----->
    // NO SIRVE EMPLEAR FOR
    i=0;
    while (i<14){
        cont=0;
        j=i+1;
        while ((m[i]==m[j]) && (j<15)) {
            cont++;
            j++;
        };

        if (cont>0)
            cout << "El elemento " << m[i] << " se repite "
                << cont << " veces" << endl;
        i=j;
    }
    cout << "-----" << endl;
    return 0;
}
```

int m[15]={3,3,3,6,6,6,7,8,8,9,10,16,16,30,32};

NO SIRVE EMPLEAR FOR

```
El elemento 3 se repite 2 veces
El elemento 6 se repite 2 veces
El elemento 8 se repite 1 veces
El elemento 16 se repite 1 veces
-----
```

<< El programa ha finalizado: codigo de salida

NO SIRVE EMPLEAR FOR

```
i=0;
while (i<14){
    cont=0;
    j=i+1;
    while ((m[i]==m[j]) && (j<15)) {
        cont++;
        j++;
    };

    if (cont>0)
        cout << "El elemento " << m[i] << " se repite "
        << cont << " veces" << endl;
    i=j;
}
cout << "-----" << endl;
return 0;
```

```
El elemento 3 se repite 2 veces
El elemento 6 se repite 2 veces
El elemento 8 se repite 1 veces
El elemento 16 se repite 1 veces
-----
```

```
<< El programa ha finalizado: codigo de salida
```

Ejemplo 13. Repetidos en arreglos sin orden previo

```
int main() {
    int i, j, cont;

    // Trabajando con arreglos sin orden previo

    int mat[15]={3,7,5,6,2,5,3,6,2,3,1,6,0,3,3};

    //
    // ----->
    //
    // ----->
    // SE PUEDEN EMPLEAR FOR
    for(i=0; i<14; i++){
        cont=0;
        for(j=i+1; j<15; j++){
            if (mat[i]== mat[j]) cont++;
            if (cont > 0)
                cout << "El elemento " << mat[i] << " se repite "
                << cont << " veces" << endl;
        }
        cout << "-----" << endl;
    }

    // PERO VEAMOS QUE EL PROBLEMA NO SE SOLUCIONA

    return 0;
}
```

```
El elemento 3 se repite 4 veces
El elemento 5 se repite 1 veces
El elemento 6 se repite 2 veces
El elemento 2 se repite 1 veces
El elemento 3 se repite 3 veces
El elemento 6 se repite 1 veces
El elemento 3 se repite 2 veces
El elemento 3 se repite 1 veces
-----
```

// ALTERNATIVA 1 - Guardar los repetidos en otro arreglo

```
***** Crear x[15] con fin en -1 *****
```

```
for (i=0;i<14;i++) {
```

```
if (!esta (mat[i], x ,fin) {
```

```
*****Búsqueda secuencial***
```

```
    c=0;
```

```
    for(j=i+1;j<15;j++)
```

```
        if (mat[i]== mat[j]) c++;
```

```
    if (c>0) cout << "el elemento "<< mat[i]<<
```

```
        " se repite "<< c << " veces"<< endl;
```

```
        insertar (mat[i],x,fin)
```

```
***** Insertar al final*****
```

```
    };    };
```

//SI EL ARREGLO TIENE 1000 ELEMENTOS Y POCOS REPETIDOS, LA BÚSQUEDA SE REALIZARÁ CASI cercana a 1000, ASÍ QUE hará aproximadamente 1.000.000 DE CONSULTAS

// ALTERNATIVA 2 - Guardar ordenado en otro arreglo

```
***** Crear x[15] con fin en -1 *****
```

```
for(i=0;i<14;i++) {  
    if (!esta(mat[i],x,fin) {          *****Busqueda binaria*****  
        c=0;  
        for(j=i+1;j<15;j++)  
            if (mat[i]== mat[j]) c++;  
        if (c>0) cout << "el elemnto " << mat[i] << " se repite "  
            << c << " veces" << endl;  
  
        insertar (mat[i],x,fin)  *** Insertar ordenado en el  
                                arreglo*****  
    };  
};
```

SI EL ARREGLO TIENE 1000 ELEMENTOS Y POCOS REPETIDOS, LA BÚSQUEDA SE REALIZARÁ Con aproximadamente 10.000 (10*1000) CONSULTAS .

ALTERNATIVA 3 – Marcar los elementos que se repiten por medio de un arreglo paralelo booleano

```
int main() {
    int i, j, cont;

    // ALTERNATIVA 3 - Marcar los elementos que se repiten por medio de
    // un arreglo paralelo booleano

    int mat[15]={3,7,5,6,2,5,3,6,2,3,1,6,0,3,3};

    bool tach[15] = {false};
    for(i=0; i<14; i++){
        if (!tach [i]) {
            cont=0;
            for(j=i+1 ;j<15 ;j++)
                if (mat[i]== mat[j]) {
                    cont++;
                    tach[j] = true;
                }
            if (cont > 0)
                cout << "El elemento " << mat[i]<< " se repite "
                << cont << " veces" << endl;
        }
    }

    // SI EL ARREGLO TIENE 1000 ELEMENTOS solo se consultan 1000

    return 0;
}
```

```
El elemento 3 se repite 4 veces
El elemento 5 se repite 1 veces
El elemento 6 se repite 2 veces
El elemento 2 se repite 1 veces
```

```
// El programa ha finalizado: adi
```

Veamos soluciones
para un problema de sueldos

Problema

Escribir un programa que permita manejar **información de empleados y los sueldos correspondientes.**

Cada empleado tiene un Número de identificación y una Categoría.

Se ingresan los datos de un empleado (Nro de identificación, Sueldo, Categoría, Cantidad de Hijos) y se debe mostrar en pantalla lo siguiente: Nro: nnn, Sueldo : kkkk, Sueldo nuevo: llll

El sueldo nuevo se calcula con un incremento según la categoría y a ello un incremento de 1% por cada hijo

Parte de main (Ejemplo 8. Problema de sueldos)

```
int nro,hijos,dni,i, j;
float sueldo,sNuevo, maxSNuevo=0;
char cat;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout << endl ;
cout << "Cuantos empleados va a procesar : ", cin >> j;
cout << "Se inicia el proceso" << endl<< endl;
for (i=0;i<j;i++) {
    ingresarDatos (dni,nro,cat,sueldo,hijos);
    if (validarDatos (dni,nro,cat,sueldo,hijos))
        mostrarSalida(dni,nro,cat,sueldo,hijos,sNuevo);
    cout << endl<<endl<<endl << endl;
    mostrarRombo(9);
    if (sNuevo>maxSNuevo) maxSNuevo=sNuevo;
};
cout<<"DATOS ESTADISTICOS:" << endl
    <<"El mayor sueldo nuevo generado es: $ "<<maxSNuevo<<endl;
return 0;
```

Ver obtención del Máximo sueldo nuevo

```
int nro,hijos,dni,i,j;
float sueldo,sNuevo, maxSNuevo=0;
char cat;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout << endl;
cout << "Cuantos empleados va a procesar : ", cin >> j;
cout << "Se inicia el proceso" << endl<< endl;
for (i=0;i<j;i++) {
    ingresarDatos (dni,nro,cat,sueldo,hijos);
    if (validarDatos (dni,nro,cat,sueldo,hijos))
        mostrarSalida(dni,nro,cat,sueldo,hijos,sNuevo);
    cout << endl<<endl<<endl << endl;
    mostrarRombo(9);
    if (sNuevo>maxSNuevo) maxSNuevo=sNuevo;
};
cout<<"DATOS ESTADISTICOS:" << endl
    <<"El mayor sueldo nuevo generado es: $ "<<maxSNuevo<<endl;
return 0;
```

← Inicializar

← Tratar cada sueldo nuevo

← Informar al final

Problema modificado

Se plantean los siguientes **requerimientos** :

3.a) La solución debe prever que se utilice en una organización que tiene varias categorías, identificadas con números (0,1,2,...n). El último valor no se conoce (si bien se sabe que es menor o igual que 100), pero el usuario lo ingresará como dato al programa.

Los porcentajes de incrementos de sueldos para cada categoría tampoco se conocen y se ingresarán como dato del programa (en cada renglón: nro de categoría – porcentaje de incremento)

3.b) La solución debe permitir generar los sueldos nuevos de todos los empleados de la organización. El número de empleados no se conoce, pero se indicará el final con número de identificación 0.

3.c) La estadística final ahora deberá informar:

- i- Indicar para cada categoría, cual es el valor total de \$ requeridos para afrontar los aumentos de los empleados de esa categoría (total de aumento pro categoría).
- ii- Cuál es el promedio de total de aumento por categoría.
- iii- Cuántas categorías tienen un total de aumentos que supera el promedio.
- iv- Indicar si hay categorías que tienen el mismo total de aumentos.

Solución 3.a)

```
//Este programa brinda apoyo para calculo e inormación  
//fue elaborado por AEED en agosto 2014- Santa Fe- Arg
```

```
const int NUM_CAT = 100;
```

```
int i, aumentos[NUM_CAT], tam;
```

```
void ingresarDatos (int& ,int& ,int& ,float& ,int& );
```

```
int validarDatos (int& ,int& ,int& ,float& ,int& );
```

```
void mostrarSalida(int,int,int,float,int,float&);
```

```
float sueldoNuevo ( int, float,int);
```

```
void mostrarClave (int);
```

```
int invertir(int, int);
```

```
void mostrarRombo (int);
```

Solución 3.a)

```
cout << endl ;  
cout << "Ingresar el nro de categorias : ", cin >> tam;  
cout << endl << "Ingresar en cada renglon cat y porcent : " << endl;  
for (i = 0; i < tam; i++)  
{  
    cin >> c;  
    cin >> porc;  
    cout << endl;  
    aumentos[c]= porc;  
}  
  
cout << "Cuantos empleados va a procesar : ", cin >> j;
```

Solución 3.a)

```
/* VALIDACION DE CATEGORIA*/  
while (!((cat>=0) && (cat <=tam-1)) && (c<=5)){  
    cout<<" Categoria erronea. Ingrese un numero entre 0 y " << tam-1<<endl ;  
    cin>> cat;  
    c++;  
};
```

```
float sueldoNuevo (int cat, float sueldo,int hijos){  
    float snuevo;  
    snuevo= sueldo *( 1 + float(aumentos[cat])/100);  
    snuevo*= 1 + 0.01*hijos;  
    return snuevo;  
}
```

Solución 3.b)

```
cout << "Se inicia el proceso" << endl<< endl;
ingresarDatos (dni,nro,cat,sueldo,hijos),
while (nro!=0)
{
    if (sueldo<minS)minS=sueldo;
    if (validarDatos (dni,nro,cat,sueldo,hijos))
        mostrarSalida(dni,nro,cat,sueldo,hijos,sNuevo);
    cout << endl<<endl<<endl << endl;
    mostrarRombo(9);

    if (sNuevo>maxSNuevo) maxSNuevo=sNuevo;
    if (sNuevo <6000){
        cantMen ++; promMen+=sNuevo;};
    cant++;
    if (sNuevo >10000) cantMDM++;
    ingresarDatos (dni,nro,cat,sueldo,hijos);
};

cout<<"DATOS ESTADISTICOS:" << endl
    <<"El mayor sueldo nuevo generado es: $ " <<maxSNuevo<<endl
```

Solución 3.c.i)

```
cout<<"DATOS ESTADISTICOS:" << endl
    <<"El mayor sueldo nuevo generado es: $ " << maxSNuevo << endl
    <<"El menor sueldo inicial ingresado es: $ " << minS << endl;
if (cantMen!=0)
    cout<< "El promedio de sueldos nuevos menores que 6000 es:"
    << promMen/cantMen << endl
    <<"El porcentaje de los sueldos nuevos mayores que 10.000 es: "
    << int ((float(cantMDM)/cant)*100) << "%";

cout<<endl << endl <<"Totales de aumentos por categorias:" << endl;
for (i = 0; i < tam; i++)
{
    cout << "Categ: " << i << "\t" << " $ " << totales[i] << endl ;
}

return 0;
```


Faltan las soluciones

3.c.ii

3.c.iii

3.c.iv

.....

A pensarlas.....

Extendamos el Problema

El contexto del Problema 2 que estuvimos solucionando ha cambiado y se plantean los siguientes **cambios** (identificamos como **Problema 4**) :

4.a) Cada empleado trabaja en uno de los 20 Sectores de la organización, por lo que al cargar los datos, también se ingresará el número del sector donde se desempeña.

4.b) La estadística final ahora deberá informar además:

i- Para un determinado sector ingresado como dato, informar cuantos empleados se ingresaron.

ii- Para cada categoría indicar cuantos empleados se ingresaron

iii-Cuál es el sector con más empleados ingresados y cual es la categoría con más empleados ingresados

iv- Para un sector ingresado como dato, indicar cual es la categoría en la que hay más empleados ingresados.

v- Para una categoría ingresada como dato, indicar cuál es el sector en el que hay más empleados ingresados

vi-Cuál es la categoría con más empleados ingresados que trabajan en los 10 primeros sectores.

vii- Indicar cual es el sector y en que categoría hay más empleados ingresados