Algoritmos y Estructuras De Datos

ESTRUCTURAS

DE DATOS

Arreglos

ESTRUCTURAS: de Control, de Programas y de Datos

Programación ESTRUCTURADA

- **Estructuras de control** — Para organizar el flujo de ejecución de las acciones del programa
- **Estructuras de programa** — Para organizar los módulos del programa
- **Estructuras de Datos** — Para organizar los datos que se van a manejar en el programa

ESTRUCTURAS DE DATOS

- **Simples o básicos:** caracteres, booleanos, enteros, 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

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)

ARREGLOS

Con frecuencia se puede tener un **conjunto de valores**, todos del **mismo tipo de datos**, que forman un **grupo lógico** o **lista** de valores.

Por ejemplo, una lista de temperaturas:

Temperaturas
95.75
83.0
97.625
72.5
86.25

Una lista simple que contiene elementos individuales del mismo tipo de datos se llama **arreglo unidimensional**.

Un **arreglo unidimensional**, el cual también se conoce como arreglo de dimensión única, es una lista de valores relacionados con el mismo tipo de datos que se almacena usando un **nombre de grupo único**.

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 **índice** o **subíndice** del arreglo y localizan la posición del elemento dentro del arreglo.

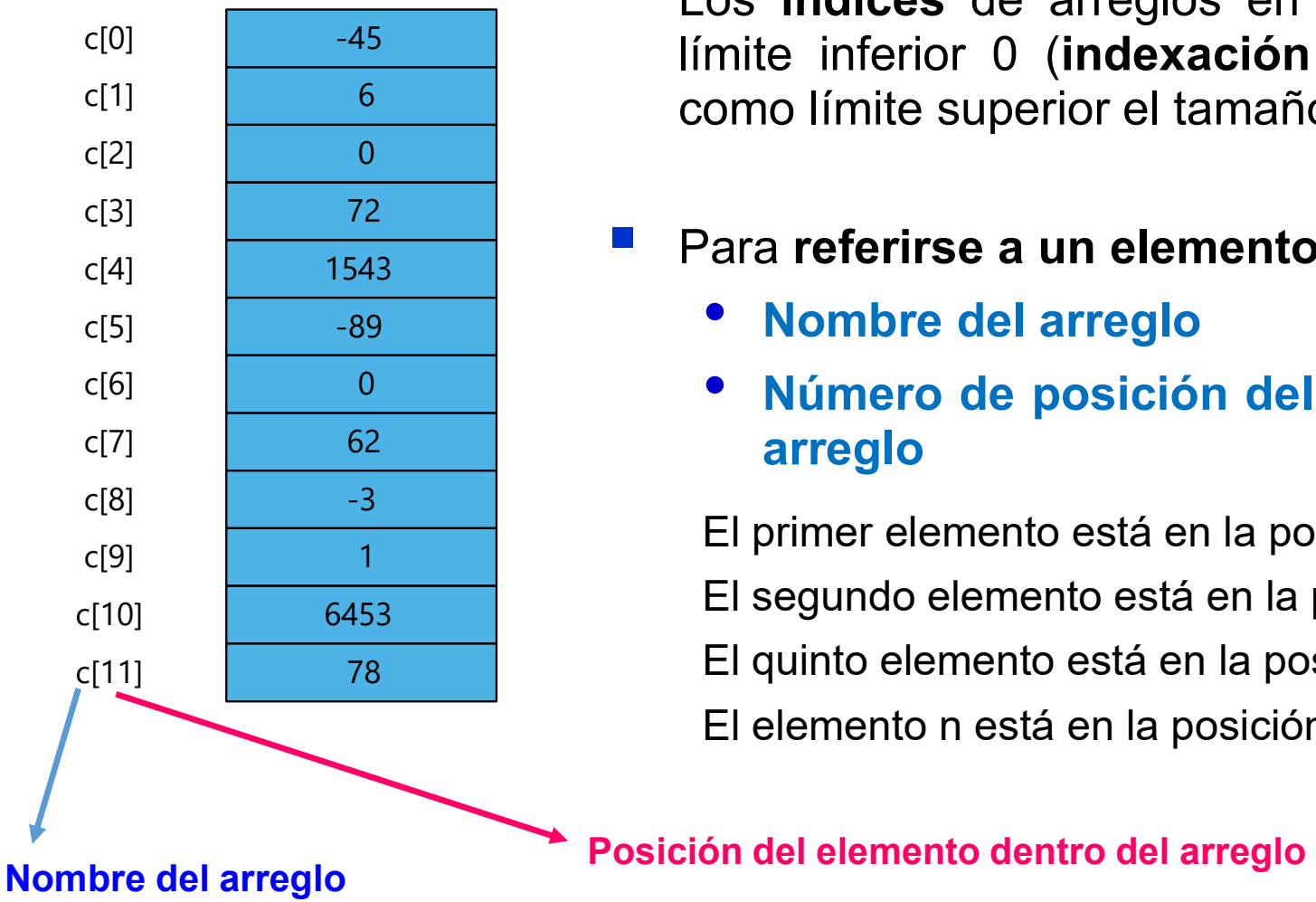
- Los elementos del arreglo se almacenan siempre en **posiciones consecutivas** de la memoria.

- Los elementos se **acceden** **en forma directa**, indicando el nombre del arreglo y el subíndice (**variables indizadas**).

- Pueden ser **unidimensionales** (se acceden con un solo índice) o **multidimensionales** (se acceden con varios índices).

Arreglos

Nombre del arreglo: Todos los elementos del arreglo tienen el mismo nombre. En el ejemplo, el nombre del arreglo es **c**.



- 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 posición del elemento dentro del arreglo**

  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]

Arreglos: Declaración

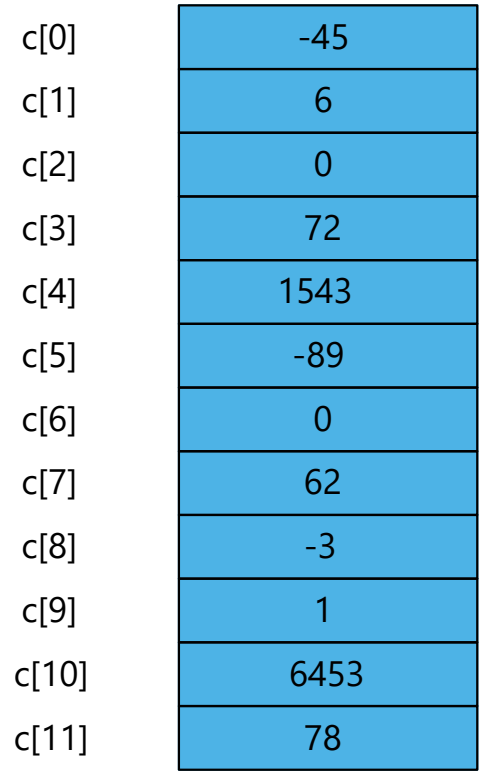
- Los arreglos se deben declarar antes de utilizarse.
- Formato de declaración:

tipo-de-dato 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]*.



Arreglos: Declaración

- El **tamaño de un arreglo** puede ser un *valor constante* o estar representado por una constante:

```
#define TAM 12      // Define una constante para el tamaño del arreglo
int c [TAM];        // Declara el arreglo
```

El tamaño de un arreglo debe ser una **constante**

- **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 elementos float
char    nombre[15];                      // Declara un array de caracteres
```

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

```
typedef  float tNotas [30];   // Define un tipo arreglo de 30 elementos float
tNotas curso;                 // Declara una variable de tipo arreglo de 30 elementos float
```

Arreglos: Almacenamiento

Un arreglo tiene tres partes:

- **La dirección** (ubicación en memoria de la primera variable indizada)
- **El tipo base del arreglo** (determina cuánta memoria –*en bytes*– 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.

¡Es responsabilidad del programador!

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};      /* Declara un array de 3 elementos */
int b [ ] = {4,6,7};         /* Declara un array de 3 elementos */
```

  - **Con asignaciones:**

```
for (i=0; i<3; i++)
        a[i]=0;
```

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

  - **Desde el teclado:**

```
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**:

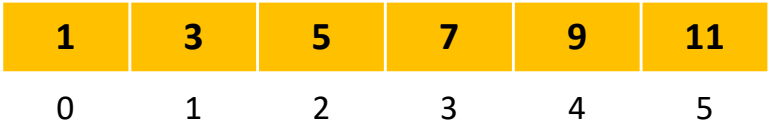
```
typedef  int  tNuevo[3];

tNuevo a = {12, 34, 45};
tNuevo b = {4, 6, 7};
```

Inicializar: Cargar un arreglo

Un arreglo, por ej: int A[6]; :

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

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



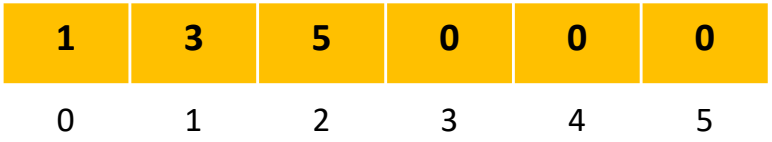
- Puede **cargarse elemento a elemento:**

```
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. Incluso pueden estar en cualquier orden.

- Pueden **cargarse menos elementos:**

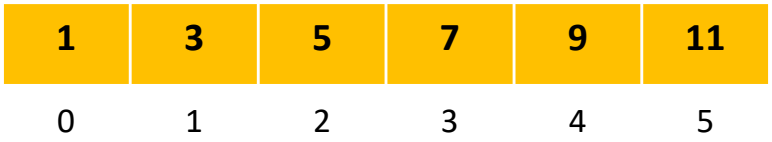
```
A[0]=1; A[1]=3; A[2]=5;
```



El resto de los elementos del arreglo serán inicializados en cero. Sólo es válido si los valores a cargar (diferentes de 0) están en posiciones consecutivas.

- 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 **cargarse desde el teclado, en el mismo orden** en que los datos 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 **cargarse desde el teclado**, pero **no necesariamente en su orden**. Para ello se debe ingresar también la posición:

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

Ingreso:

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

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

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 haciendo:

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

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

```
int N=8;
for (i=0;  i<N; i++) {
    cin >> k;
    if ((k>=0) && (k<=5))  A[k]++ ;
}
```

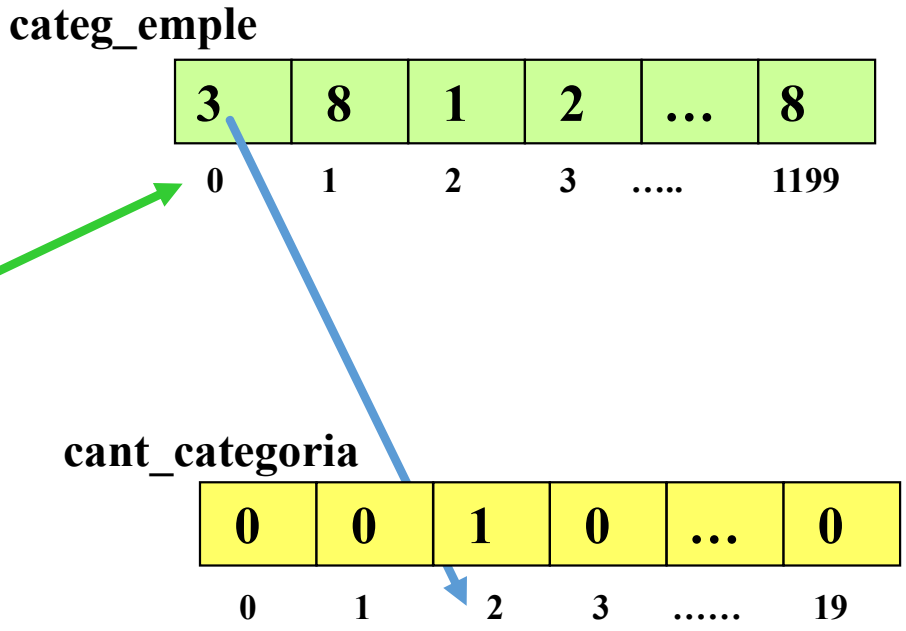
Ingreso: 1 7 2 0 5 2 8 2

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]={0};
 int i;

for (i=0; i<1200; i++)
     cin >> categ_emple[i];


 for (i=0; i<1200; i++){
     cant_categoria [categ_emple [i]-1 ]  ++;
 }
```



Arreglos: Almacenamiento en memoria

- **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**.

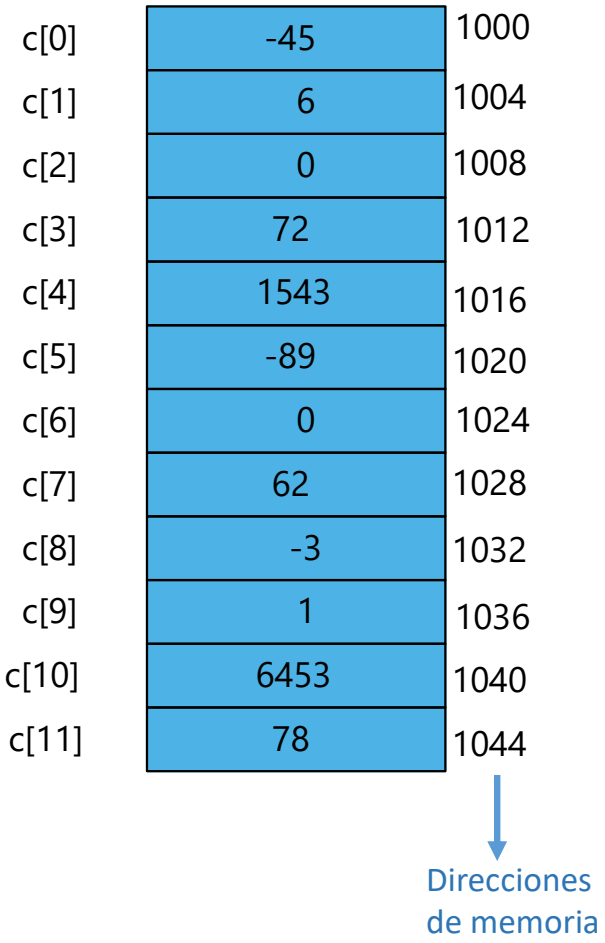
- **Espacio ocupado** por c: 48 bytes, pues posee 12 elementos de tamaño t = 4 bytes.

- **Direcciones:**

&c[0] = 1000 = D

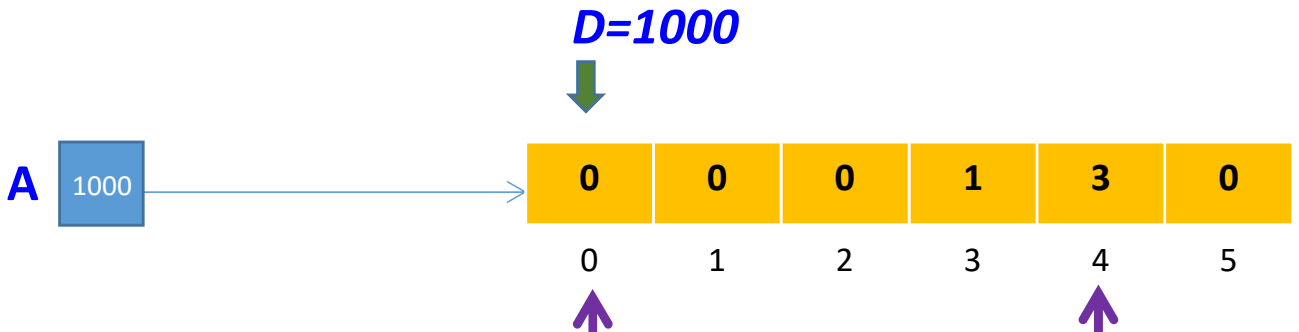
&c[i] = D + i * t ⇒ &c[4] = 1000 + 4 * 4 = 1016

&c[i] = D + i * t ⇒ &c[10] = 1000 + 10 * 4 = 1040



Arreglos en C++: Acceso directo

int A[6] ;



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 $\Rightarrow$ &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: codigo
```

Arreglos Lineales: Operaciones

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

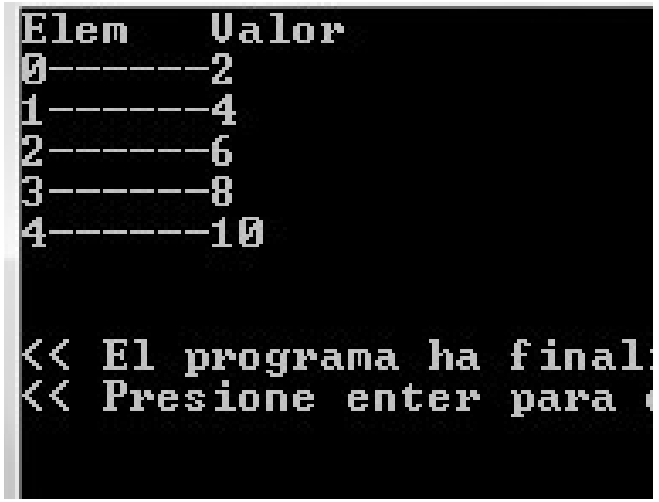
- ⇢ Imprimir los elementos de un arreglo
- ⇢ Hallar el promedio de los datos de un arreglo
- ⇢ Insertar un elemento en un arreglo
- ⇢ Borrar un elemento de un arreglo (conociendo el índice o el valor)
- ⇢ Borrar un rango
- ⇢ Buscar un elemento en un arreglo.

Operación: Imprimir elementos

- **Ejemplo:** **Imprimir los elementos de un arreglo**

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

```
#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;
}
```



Arreglos: 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 arreglo** (subíndice del último elemento cargado).
- **Tamaño Físico:** **cantidad máxima** de elementos que tiene un arreglo.
- **Tamaño Real:** **cantidad exacta** de valores almacenados dentro del arreglo.
  - También se lo conoce como tamaño **actual**, tamaño **lógico** o **tope lógico**.
  - Puede ser **menor o igual al tamaño físico** (pero nunca mayor).

En arreglos subocupados, los recorridos del arreglo deben realizarse teniendo como tope al tamaño real.

Operación: Hallar el promedio de los elementos

- **Ejemplo:** 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 tamanio: ";
    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;
}
```

Tamaño físico

Tamaño lógico

```
Ingrese el tamanio:  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
```

Pasaje de Arreglos a Funciones

Ejemplo de prototipo: `void funcion( int a[], int tam );`

Si se inserta un número dentro de los corchetes, el compilador lo ignorará.

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

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

Un parámetro de arreglo no efectúa una copia del contenido del arreglo, sino que copia la referencia a la dirección de memoria en la que empieza el arreglo.

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

Pasaje de Arreglos a Funciones

Supongamos el prototipo:


```
void funcion( int a[], int tam );
```

Si tenemos las siguientes declaraciones:

```
int a[5], numero_de_elementos = 5;
```

Una posible llamada a la función **`funcion`** podría ser:

```
funcion(a, numero_de_elementos);
```

El argumento **`a`** dado en la llamada a la función **`funcion`** se da sin corchetes ni expresión de índice.

El argumento arreglo **no le indica a la función qué tamaño tiene el arreglo**. Por ello, resulta indispensable **incluir siempre otro argumento de tipo int** que indique dicho tamaño.

Operación: Hallar el promedio de los elementos

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

float promedio(int a[], int tl);

int main(){
    int arreg[TamF];

    int tamL,suma=0;

    cout << "Ingrese el tamanio logico (tam fisico=10): ";
    cin >> tamL;

    if (tamL > 0 and tamL <= TamF){
        for (int i=0; i<tamL; i++){
            cout << endl << "Ingrese el elemento " << i << ": ";
            cin >> arreg[i];
        }
        cout << endl << "Promedio de elementos: " << promedio(arreg,tamL);
    }
    return 0;
}

float promedio(int a[], int tl){
    int suma=0;
    for(int i=0; i<tl; i++)
        suma += a[i];
    return (float)suma/tl;
}
```

```
Ingrese el tamanio:  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
```

Parámetro de arreglo constante en funciones

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

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.

Un parámetro de arreglo modificado con **const** es un **parámetro de arreglo constante**.

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

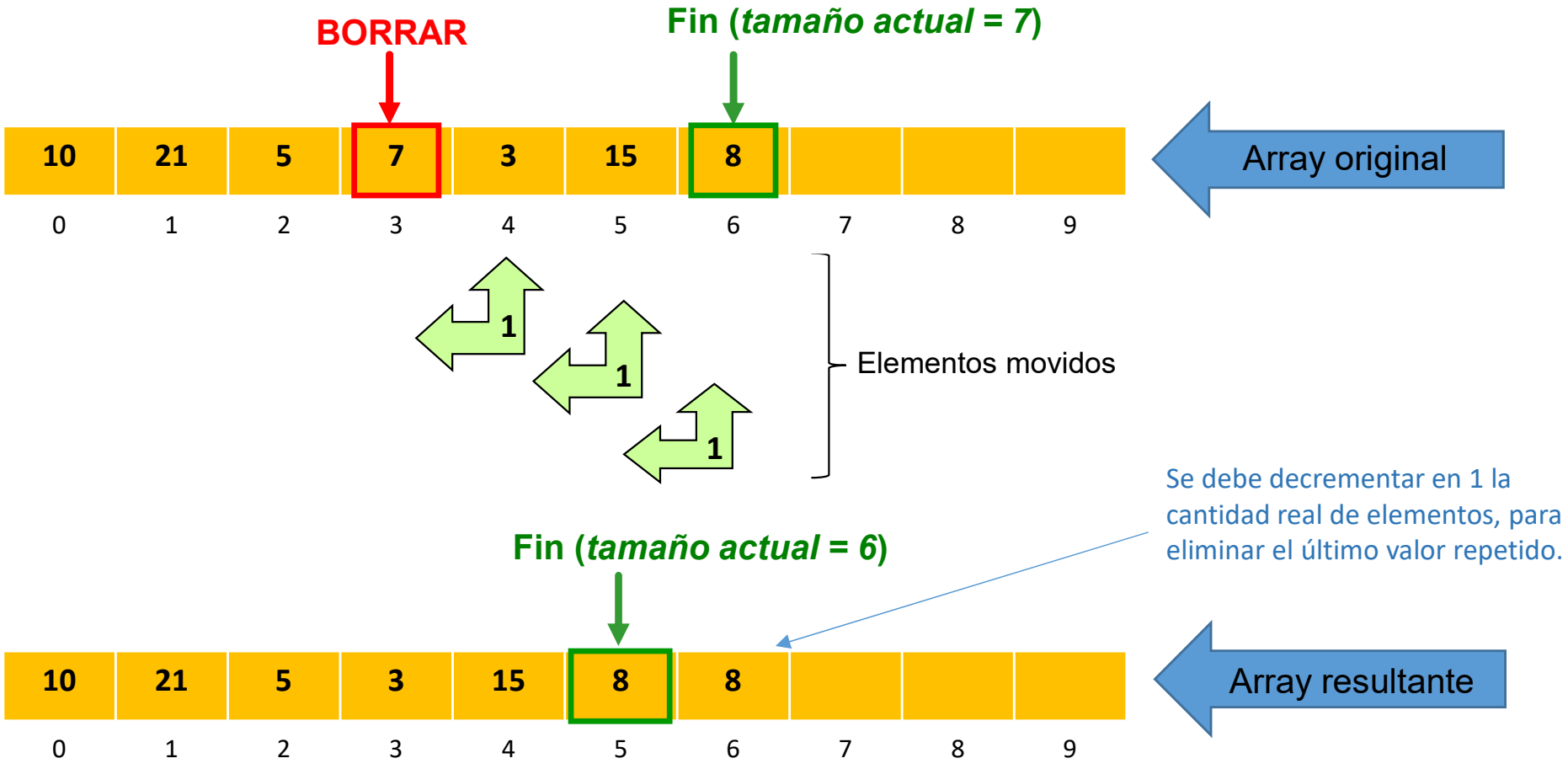
La siguiente sentencia será detectada como error:

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

Operación: Borrar un elemento de un arreglo

Borrar un elemento de un arreglo: implica correr los elementos que estén a la derecha del elemento a borrar, 1 posición hacia la izquierda, comenzando por el más cercano a dicho elemento. Y además, decrementar en 1 la cantidad real (tamaño lógico o actual) de elementos.

- Ejemplo: **Tamaño físico = 10; Tamaño actual = 7; Borrar el 4to elemento**

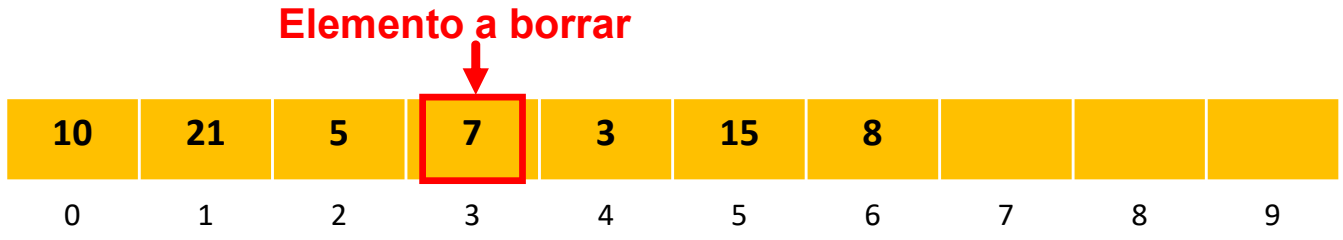


Operación: Borrar un elemento

Posición del elemento a borrar

Tamaño real

```
for (int i=pos; i<Fin-1; i++) {
        vec[ i ] = vec[ i+1 ];
}
Fin--;
```



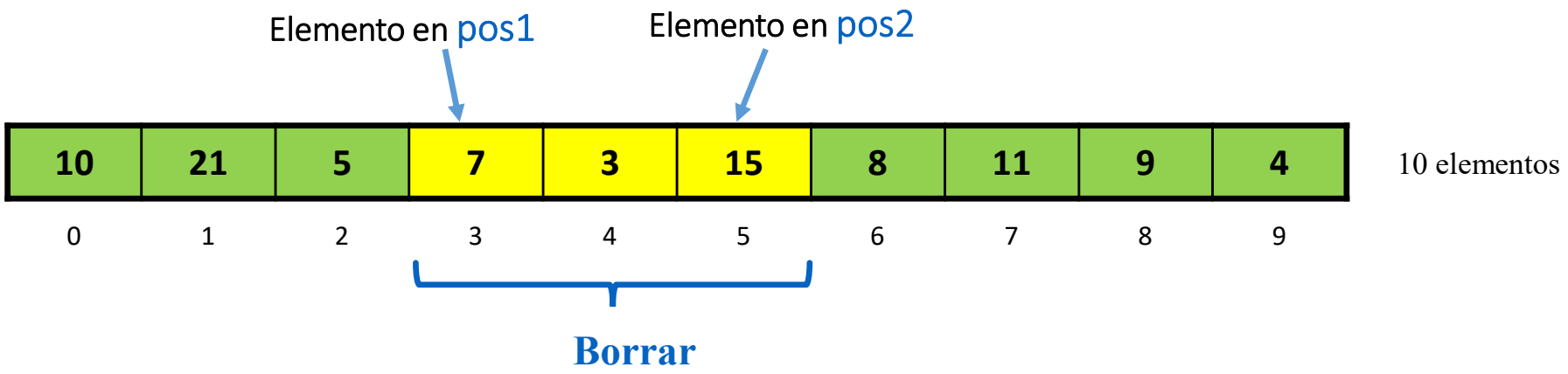
Operación: Borrar un elemento de un arreglo conociendo su posición

```
Ingrese el tamanio: 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 posicion del elemento a borrar: 2

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

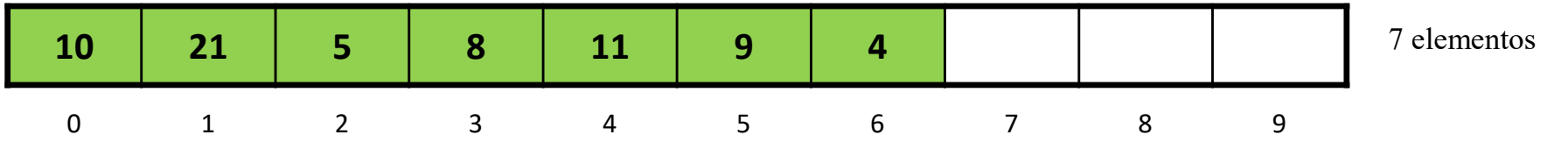
```
# define Tam 10

int main(){
    int c[Tam];
    int Fin, i, pos;

    cout << "Ingrese el tamanio: ";
    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 posicion del elemento a borrar: ";
        cin >> pos;
        for (i=pos; 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;
}
```

Operación: Borrar un rango

Borrar un rango: Se debe borrar desde pos1 hasta pos2. Por lo tanto, los elementos a la derecha de pos2, y hasta el fin del array, deberán correrse (pos2-pos1+1) posiciones a la izquierda. Luego, decrementar el tamaño lógico en la misma cantidad de elementos eliminados.



pos2 - pos1 + 1 = 5 - 3 + 1 = 3 posiciones se deben correr



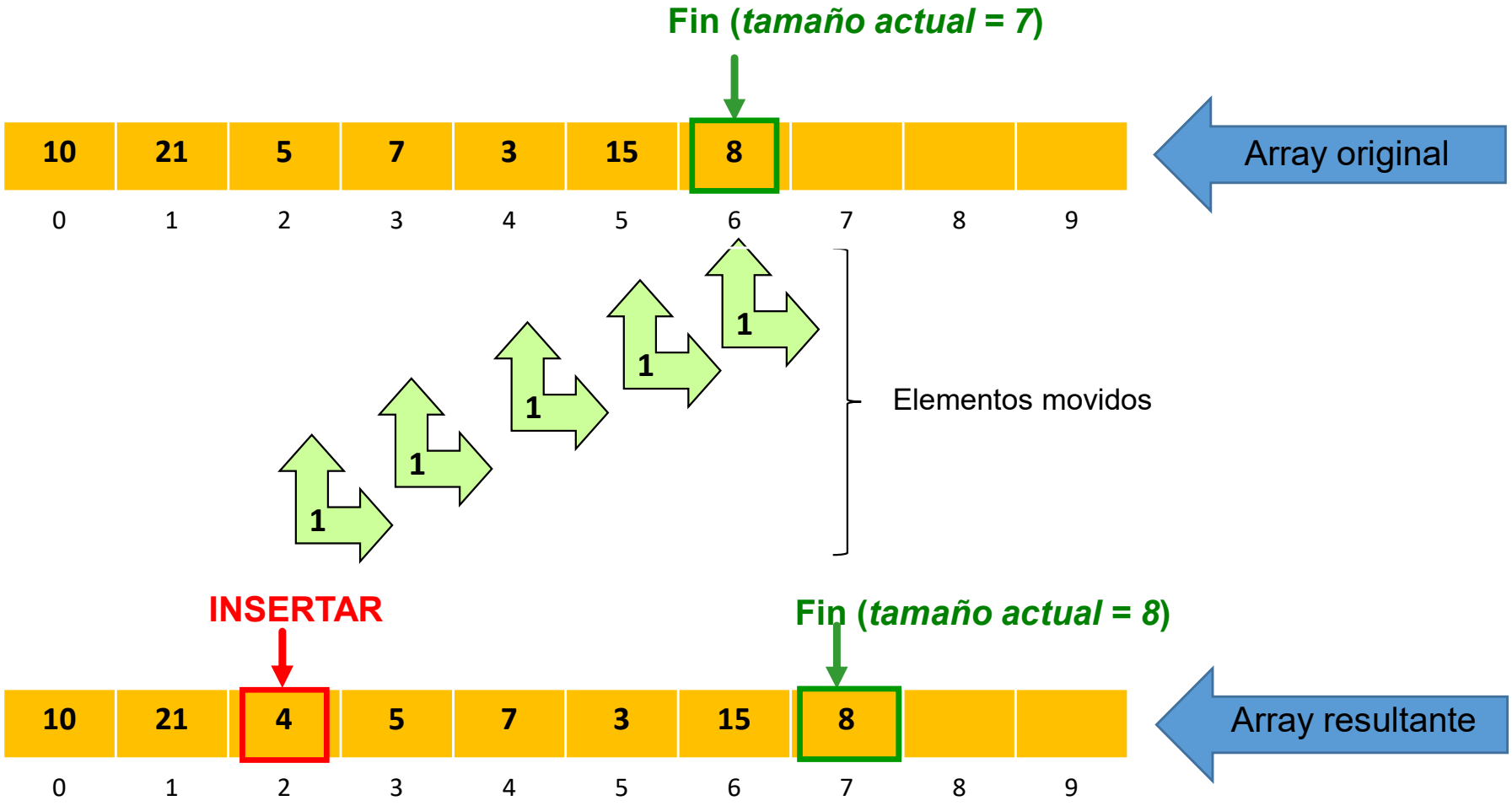
Operación: Borrar un rango

```
int cant = pos2-pos1+1;

for (int j=pos2+1; j<=Fin-1; j++) {

      vec[ j-cant ] = vec[ j ];
 }

Fin = Fin-cant;
```

Operación: Insertar un elemento de un arreglo

Implica correr los elementos 1 posición hacia la derecha, a partir de la última posición y hasta la posición en la que se va a insertar. Y además, incrementar en 1 la cantidad real (tamaño lógico o actual) de elementos.

- Ejemplo: **Tamaño físico = 10; Tamaño actual = 7; Insertar 4 en posición 2**



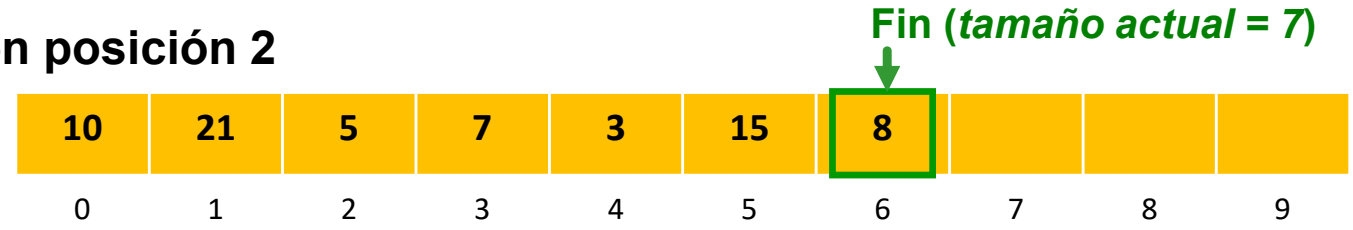
Operación: Insertar un elemento

Tamaño real

Posición donde se insertará el elemento

```
for (int i=TL-1; i>=pos; i--) {
        vec[ i+1 ] = vec[ i ];
}
vec[ pos ]=valor; //se inserta el elemento
TL++;
```

Insertar 4 en posición 2



Operación: Insertar un elemento en un arreglo

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

void cargarvector(int v[], int TL);
void mostrarvector(const int v[], int TL);
void insertar(int v[], int & TL, int valor, int pos);

int main(){
    int TL, valor, posicion;
    int v[TF];

    do{
        cout << "Ingrese tamanio (hasta 10): ";
        cin >> TL;
    } while(TL <= 0 or TL > TF);

    cargarvector(v, TL);
    mostrarvector(v, TL);

    cout << "Indique el valor a insertar: ";
    cin >> valor;

    do{
        cout << "Indique la posicion donde insertar: ";
        cin >> posicion;
    } while(posicion < 0 or posicion > TL);

    insertar(v, TL, valor, posicion);
    mostrarvector(v, TL);

    return 0;
}
```

```
Ingrese tamanio (hasta 10): 5
Ingrese los elementos del vector:
1 2 3 4 5
Vector:
1 2 3 4 5
TL=5

Indique el valor a insertar: 8
Indique la posicion donde insertar: 2
Vector:
1 2 8 3 4 5
TL=6
```

Operación: Insertar un elemento en un arreglo

```
void cargarvector(int x[], int Fin){
    cout << "Ingrese los elementos del vector:" << endl;
    for (int i=0; i<Fin; i++)
        cin >> x[i];
}

void mostrarvector(const int x[], int Fin){
    cout << "Vector:" << endl;
    for (int i=0; i<Fin; i++)
        cout << x[i] << " ";
    cout << endl;
    cout << "TL=" << Fin << endl << endl;
}

void insertar(int x[], int & Fin, int valor, int pos){
    for (int i=Fin-1; i>=pos; i--)
        x[i+1] = x[i];
    x[pos]=valor;
    Fin++;
}
```

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<=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:

```
int i=0;

while (i<N and A[i]!=X){
    i++;
}

if(i<N)
    cout << "X encontrado en posicion " << i;
else
    cout << "X no encontrado";
```

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

Operación: Buscar un elemento

Alternativa usando for → *Opción no válida para la cátedra!*

- **Ejemplo:** Buscar un elemento en un arreglo usando Búsqueda Lineal

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

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

¡Cuidado! 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<=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 qué 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)
    cout << "X no encontrado";
else
    cout << "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:** **Buscar un elemento en un arreglo usando Búsqueda Binaria**

```
//Esta función devuelve la ubicación de clave en el vector
//Se devuelve -1 si no se encuentra el valor
int busquedaBinaria(int vec[], int tamanio, int clave){
    int izquierdo, derecho, puntomedio;
    bool encontrado = false;
    izquierdo = 0;
    derecho = tamanio-1;

    while (izquierdo <= derecho and !encontrado){
        puntomedio = (int)((izquierdo + derecho) / 2);
        if (clave == vec[puntomedio])
            encontrado = true;
        else if (clave > vec[puntomedio])
            izquierdo = puntomedio+1;
        else
            derecho = puntomedio-1;
    }

    if (encontrado)
        return puntomedio;
    else
        return -1;
}
```

Para profundizar en el tema:

Libro:
Resolución de problemas con C++ - Savitch
Capítulo 10

Libro:
C++ para ingeniería y ciencias - Gary J. Bronson
Capítulo 11