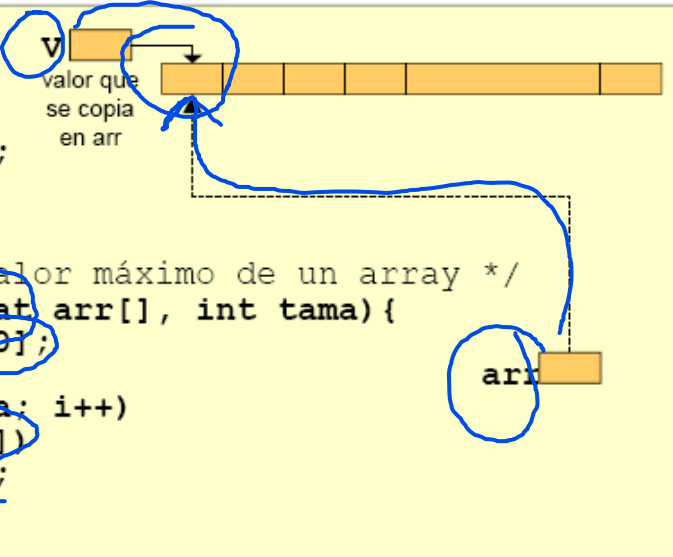


Arreglos unidimensionales como argumentos de funciones:

- Si el argumento es un array se pasa un puntero al primer elemento del array.
- El paso de un array SIEMPRE tiene un comportamiento como si fuera un paso por REFERENCIA:
 - NO se puede modificar la dirección del array.
 - La función SI puede **modificar** los elementos del array.
- Por tanto, NO es necesario poner el & si se quiere modificar un array dentro de una función. Si se pone da ERROR.
- Ejemplo: PROTOTIPO de función que usa un array:

```
float maximo(float arr[], int tama);
```

```
int main() {  
    float V[50], m;  
    .....  
    m=maximo(V, 50);  
    .....  
}  
  
/* Devuelve el valor máximo de un array */  
float maximo(float arr[], int tama){  
    float max=arr[0];  
    int i;  
    for(i=1; i<tama; i++){  
        if(max<arr[i])  
            max=arr[i];  
    }  
    return max;  
}
```



- No se reserva espacio para el array (arr en el ejemplo) en la función: el parámetro es un puntero al espacio que fue previamente reservado en otro sitio (V en el ejemplo).
- Un parámetro array es compatible con arrays de cualquier tamaño.

Ejercicio: Hacer funcionar el programa anterior y comprobar que el arreglo "v" se esta enviando por referencia.

Ejemplo:

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
```

```
void modifica(int arreglo[], int tamano);
```

```
int main(){
    const int tam_arr=5; // Calificador const declara tam como variable
    //constante
```

```
    int MiArreglo[tam_arr]={5,10,15,20,25};
    int i;
```

```
    modifica(MiArreglo, tam_arr);
```

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

```
    {
        cout<<"elemento "<<i<<"es "<<MiArreglo[i]<<endl;
    }
```

```
    cout<<"presiona enter para terminar"<<endl;
```

```
    getch();
```

```
    return 0;
}
```

```
void modifica(int arreglo[], int tam_arr)
{
    for (int i=0;i<tam_arr;++i)
        arreglo[i]*=2;
}
```

- Las variables constantes deben inicializarse mediante una expresión constante cuando se declaran y no se pueden modificar posteriormente
- Esas variables constantes también se llaman **constantes con nombres** o **variables de solo lectura**

Pregunta: ¿que hace el programa anterior?

Un ejemplo más de uso de variable constante:

// Uso apropiado de una variable constante inicializada.

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    const int x = 7; // variable constante inicializada
```

```
    cout << "El valor de la variable constante x es: "  
        << x << endl;
```

```
    return 0; // indica terminación exitosa
```

```
} // fin de main
```

Un ejemplo de uso incorrecto de la variable constante:

```
int main()
```

```
{
```

```
    const int x; // Error: x debe inicializarse
```

```
        x = 7;          // Error: no se puede modificar una //  
        //variable     const
```

```
    return 0; // indica terminación "exitosa"
```

```
} // fin de main
```

También se puede pasar a una función el valor de un elemento del array, sólo se pasará el valor del elemento, no se puede modificar el elemento original.

FuncionInvocada (array [elemento]);

Pregunta: Porque no se modifica el elemento original?

- Si no queremos que la función cambie el arreglo:
Incluir el calificador *const* dentro del prototipo y definición de la función

tipoRetorno nombreFuncion (const tipoArreglo nombre []);

- Cuando la función especifica un parámetro de arreglo que esta precedido por el calificador *const*, los elementos del arreglo se vuelven constantes en el cuerpo de la función
- Cualquier intento de modificar uno de sus elementos en el cuerpo de la función provoca un error de compilación

Ejemplo de intento de modificar un arreglo con *const*.

```
#include <iostream>
```

```
void intentaModificarArreglo( const int [ ] ); // prototipo de la  
función
```

```
int main()
{
    int a[ ] = { 10, 20, 30 };

    intentaModificarArreglo( a );

    cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';

    return 0; // indica terminación exitosa

} // fin de main
```

```
void intentaModificarArreglo( const int b[] )
{
    b[ 0 ] /= 2; // error
    b[ 1 ] /= 2; // error
    b[ 2 ] /= 2; // error

} // fin de la función intentaModificarArreglo
```

Ejercicio: Hacer un programa que lea diez valores enteros en un arreglo desde el teclado y calcule y muestre: la suma, el valor promedio, el mayor y el menor. Implementan para la suma, promedio, calculo de mayor y menor valores cuatro funciones diferentes y como parámetro les envían el vector.

Tarea para practicar en la casa:

1. Hacer un programa que lea diez valores enteros en un arreglo y los muestre en pantalla. Después que los ordene de menor a mayor y los vuelva a mostrar. Y finalmente que los ordene de mayor a menor y los muestre por tercera vez. Para ordenar la lista usar una función que implemente el método de la burbuja y que tenga como parámetro de entrada el tipo de ordenación, de mayor a menor o de menor a mayor. Para el arreglo usar una variable global.