



# Capítulo 3: Estructuras y Algoritmos de resolución de problemas

Profesores: Johan Baldeón – David Allasi

**INF237 – Lenguaje de Programación Orientada a Objetos**

# Agenda

- Arreglos
  - Algoritmos de ordenamiento y búsqueda
  - Arreglos de caracteres
  - Arreglos bidimensionales
- Funciones
  - Prototipos
  - Llamadas a funciones
    - Parámetros por valor
    - Parámetros por referencia
- Arreglos y Funciones

# Arreglos

# Introducción

- Existen situaciones y/o problemas que no se pueden resolver directamente con variables simples dado que se necesita manejar grandes cantidades de datos.
- Si se resuelve con variables simples su solución sería tan engorrosa de escribir que no valdría la pena resolverlos computacionalmente.
- Para resolver estas situaciones los lenguajes de programación de alto nivel como el C++, tienen una facilidad conocida como ARREGLOS.

# Definición

- Un arreglo es una facilidad del lenguaje que permite manejar una gran cantidad de datos del mismo tipo bajo un mismo nombre o identificador.
- Un vector es un arreglo unidimensional.
- Una matriz es un arreglo bidimensional.
- Cada uno de los datos que se maneja dentro del arreglo se conoce como “elemento”.

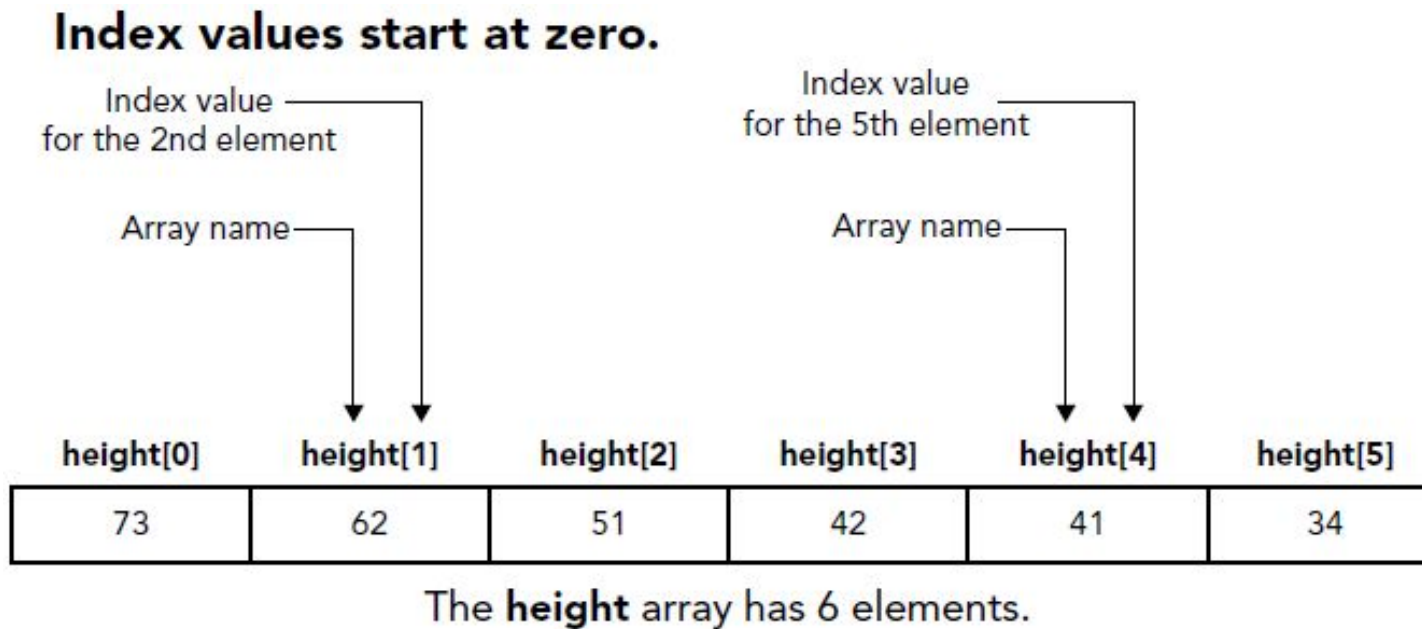
# Definición

- El tamaño del arreglo está determinado por la cantidad máxima de datos que puede almacenar.
- Es una entidad estática porque el tamaño es fijo a lo largo de todo el programa.
- Los arreglos de tamaño variable son los punteros (arreglos dinámicos).
- Los elementos de un arreglo se almacenan en un bloque contiguo de memoria.
- Por ello la cantidad de memoria necesaria para almacenar un elemento esta determinada por su tipo de dato.



# Definición

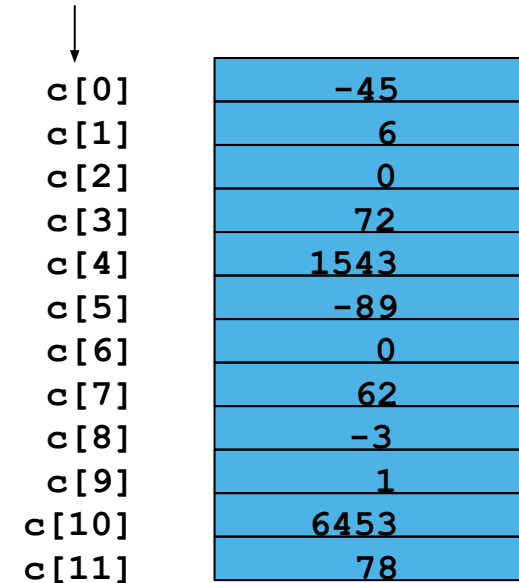
- Tamaño del arreglo = número máximo de elementos.
- Cada elemento es identificado con una *posición* dentro del arreglo o *índice*.



# Reforzando

- Arreglo, es un grupo de posiciones consecutivos de memoria, compartiendo el mismo nombre y tipo.
- Para referirnos a un elemento se debe especificar: el nombre del arreglo y la posición numérica.
- Sintaxis: `nombre_arreglo[posición_numerica]`
- El primer elemento está en la posición 0
- Un arreglo de n elementos llamado c: `c[0]`, `c[1]`, ... , `c[n-1]`

Nombre del arreglo c  
(Notar todos los elementos de este arreglo tienen el mismo nombre)



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

Posición numérica del elemento en el arreglo c



# Reforzando

- Los elementos de un arreglo se utilizan en las expresiones de C++ como cualquier otra variable:

```
c[0] = 3;  
cout << c[0]
```

- Los índices son números enteros positivos.

```
c[y-2] == c[3] == c[x];
```

- Para trabajar con los elementos puede ser: uno a uno o por medio de bucles (*for*, *while*, *do-while*).

# Declaración de Arreglos

Para declarar arreglos, se debe especificar:

- El nombre
- El tipo de arreglo (puede ser cualquier tipo de dato)
- El número de elementos

Tipo Nombre\_de\_Arreglo[Nro\_de\_Elementos];

- Ejemplos:

```
int c[10];  
float miArreglo[3284];
```

# Inicialización de Arreglos

Inicialización: `int n[5] = {1, 2, 3, 4, 5}`

- Si no se colocan todos los inicializadores, los demás a la derecha se ponen en 0:

`int n[5] = {1};`

- Si se asignan más inicializadores que el tamaño especificado, se genera un error de sintaxis.
- Si se omite el tamaño, el número de los inicializadores lo determina:

`int n[ ] = {1, 2, 3, 4, 5};`

Como hay 5 inicializadores, entonces es un arreglo de 5 elementos.

# Inicialización de Arreglos

- Si no se inicializa el arreglo, sus elementos toman valores no deseados:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int value[5] = { 1,2,3 };
```

```
    int junk[5];
```

```
    cout << endl;
```

```
    for (int i = 0; i < 5; i++)
```

```
        cout << value[i] << " ";
```

```
    cout << endl;
```

```
    for (int i = 0; i < 5; i++)
```

```
        cout << junk[i] << " ";
```

```
    return 0;
```

```
}
```

```
1 2 3 0 0
```

```
-858993460 -858993460 -858993460 -858993460 -858993460
```

# Aplicación: Ordenando Datos

- Ordenar datos es importante en computación, porque permite: encontrar datos más rápidamente y hacer cálculos estadísticos, como la moda.
- Existen múltiples algoritmos de ordenamiento de datos:
  - Burbuja (bubblesort)
  - Por selección
  - Por inserción
  - Shell
  - Rápido (quicksort)
  - Bin-sort
  - Radix-sort

# Método de la Burbuja

- Ordenar por Burbuja (sinking sort)
  - Varias pasadas a través del arreglo
  - Pares sucesivos son comparados
  - Si el orden es creciente (idénticos), no se cambia
  - Si el orden es decreciente, los elementos no se intercambian
  - Se repite el procedimiento
- Ejemplo:
  - Original:     3 4 2 6 7
  - Pasada 1:     3 2 4 6 7
  - Pasada 2:     2 3 4 6 7
  - Los elementos pequeños flotan hacia arriba



# Implementación en C++

```
void Burbuja(int a[],int tam)
{
    int pass,j,aux;

    for(pass=1; pass<=tam-1; pass++)
        for(j=0; j<=tam-2; j++)
            if(a[j] > a[j+1]){
                aux = a[j];
                a[j]= a[j+1];
                a[j+1] = aux;
            }
}
```

# Búsqueda en arreglos: Búsqueda lineal

- Buscar en un arreglo un valor clave
- Búsqueda lineal
  - Simple
  - Compara cada elemento del arreglo con valor clave
  - Útil para pequeños arreglos sin ordenar

# Búsqueda en arreglos: Búsqueda binaria

- Para arreglos ordenados
- Compara el elemento del medio con la clave:
  - Si son iguales, se encontró el valor clave
  - Si la **clave** < **medio**, se busca en la primera mitad del arreglo
  - Si la **clave** > **medio**, se busca en la otra mitad
  - Se repite sucesivamente
- Es muy rápido, máximo en  $n$  pasos
  - Donde  $2^n > \text{número de elementos}$
  - 30 elementos de arreglo toma a lo mucho 5 pasos:  $2^5 > 30$

# Otra situación especial

- Se necesita manipular texto: palabras, frases, oraciones
- Solo tenemos los datos de tipo char
- La solución es utilizar arreglos de caracteres (conocidos como cadena de caracteres)

# Arreglo de Caracteres

- La palabra “juan” se representa como un arreglo estático de caracteres
- Los arreglos de caracteres se inicializan:  

```
char palabra[] = “juan”;
```
- El caracter Nulo ‘\0’ indica fin de la cadena
- Luego, palabra[] tiene 5 elementos. Esto es equivalente a:  

```
char palabra[] = {‘j’, ‘u’, ‘a’, ‘n’, ‘\0’};
```

# Arreglo de Caracteres

- Se puede acceder a los caracteres en forma individual:  
palabra[3] es el caracter 'n'
- El nombre del arreglo es la dirección del arreglo, así que & no es necesario para scanf: `scanf( "%s", palabra );`
- Lee caracteres hasta encontrar un espacio en blanco
- Puede escribir más del fin del arreglo, por lo que se debe tener cuidado



# Arreglo de Caracteres

```
#include <iostream>

using namespace std;

int main()
{
    char string1[20], string2[] = "string literal";

    int i;

    cout << "Enter a string: ";
    cin >> string1;
    cout << "string1 is: " << string1 << endl;
    cout << "string2 is: " << string2 << endl;

    cout << "string1 with spaces between characters is: ";
    for (int i = 0; string1[i] != '\0'; i++)
        cout << string1[i] << " ";

    return 0;
}
```

Enter a string: Hello there  
string1 is: Hello  
string2 is: string literal  
string1 with spaces between characters is: H e l l o

# Arreglos Bidimensionales

- Son como matrices: se especifica la fila y luego la columna.
- Tablas en filas y columnas (arreglo de m por n).
- Por lo tanto, los arreglos tienen dos índices: uno para la fila y otro para la columna.
- Los elementos del arreglo bidimensional son almacenados en bloques contiguos de memoria.

# Arreglos Bidimensionales

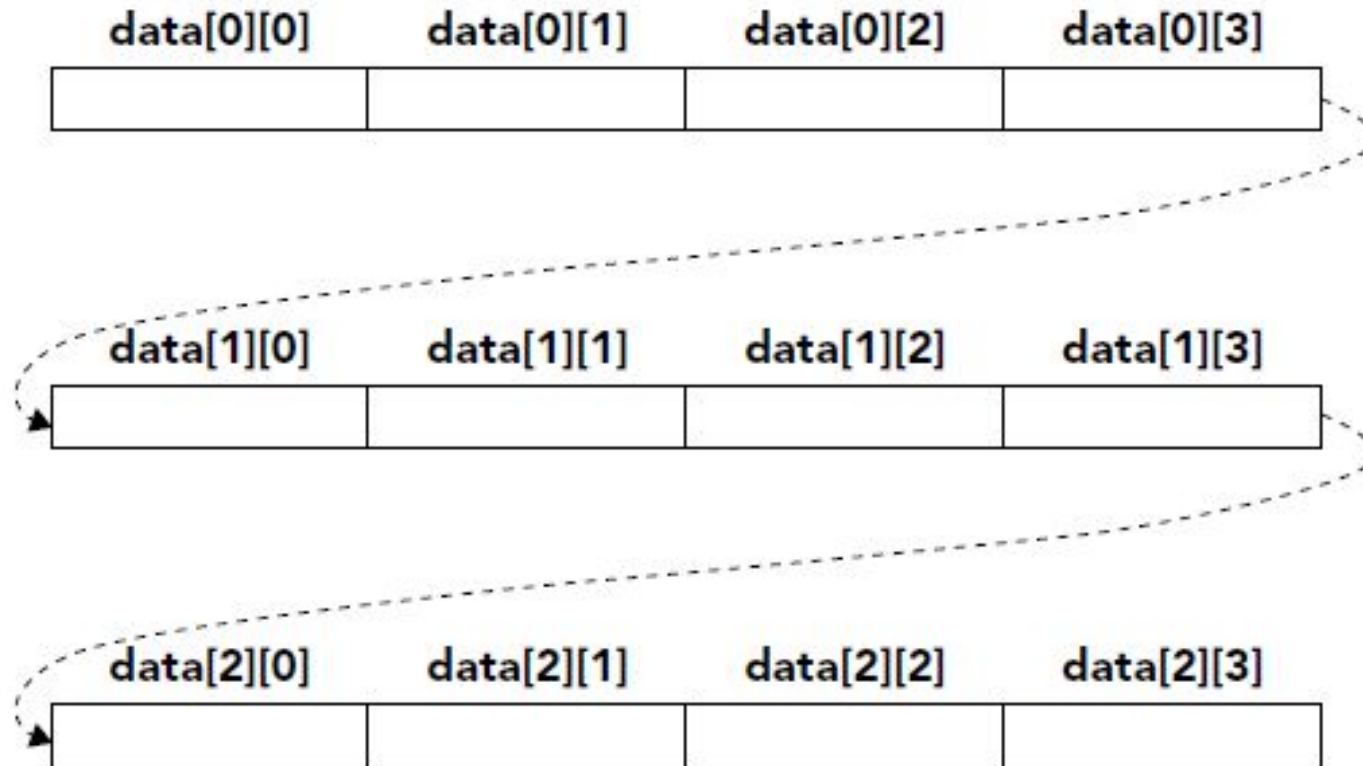
	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Fila 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Fila 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Nombre de Arreglo

Índice de fila

Índice de columna

# Arreglos Bidimensionales



Array elements are stored in contiguous locations in memory.

# Arreglos Bidimensionales

- Inicialización: `int b[2][2] = { {1, 2}, {3, 4} };`
- Inicializadores se agrupan por filas entre llaves.
- Elementos no especificados se ponen a cero  
`int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`
- Referenciando elementos, se especifica fila, luego columna:  
`printf( "%d", b[ 0 ][ 1 ] );`

# Ejercicio

Implementar un programa en C++ que lea desde la consola las 4 notas de exámenes para 3 alumnos. El programa calculará el promedio de todas las notas, la mínima y la máxima nota.



# Solución

```
#include "stdio.h"
//Definición de constantes
const int ALUMNOS = 3;
const int NOTAS = 4;
//Declaración de funciones
int ObtenerMaximo(int[ALUMNOS][NOTAS]);
int ObtenerMinimo(int[ALUMNOS][NOTAS]);
void ObtenerPromedio(int[ALUMNOS][NOTAS]);
void ImprimirMatriz(int[ALUMNOS][NOTAS]);
void LeerMatriz(int[ALUMNOS][NOTAS]);
int main()
{
    int notasAlumnos[ALUMNOS][NOTAS] = { {0,0,0,0},{0,0,0,0},{0,0,0,0} };
    int min = 0, max = 0;
    double promedio = 0;
    //invocación a funciones
    LeerMatriz(notasAlumnos);
    printf("\nLa matriz es: \n");
    ImprimirMatriz(notasAlumnos);
    min = ObtenerMinimo(notasAlumnos);
    printf_s("\n\nMinima nota: %d \n", min);
    max = ObtenerMaximo(notasAlumnos);
    printf_s("Maxima nota: %d \n", max);
    ObtenerPromedio(notasAlumnos);
    scanf_s("%d");
    return 0;
}
```

# Solución

```
void LeerMatriz(int notasAlumnos[ALUMNOS][NOTAS])
{
    int i, j;
    for (i = 0; i < ALUMNOS; i++)
    {
        printf("Ingrese las notas del alumno %d \n", i + 1);
        for (j = 0; j < NOTAS; j++)
        {
            printf("Ingrese la nota [%d]: ", j + 1);
            scanf_s("%d", &notasAlumnos[i][j]);
        }
        printf("\n");
    }
}

void ImprimirMatriz(int notasAlumnos[ALUMNOS][NOTAS])
{
    int i, j;
    printf("                [1]    [2]    [3]    [4]");
    for (i = 0; i < ALUMNOS; i++)
    {
        printf("\n Notas de Alumno[%d] ", i + 1);
        for (j = 0; j < NOTAS; j++)
            printf("%-7d", notasAlumnos[i][j]);
    }
}
```

# Solución

```
int ObtenerMaximo(int notasAlumnos[ALUMNOS][NOTAS])
{
    int i, j, max = 0;
    for (i = 0; i < ALUMNOS; i++)
        for (j = 0; j < NOTAS; j++)
            if (notasAlumnos[i][j] > max)
                max = notasAlumnos[i][j];
    return max;
}

int ObtenerMinimo(int notasAlumnos[ALUMNOS][NOTAS])
{
    int i, j, min = 20;
    for (i = 0; i < ALUMNOS; i++)
        for (j = 0; j < NOTAS; j++)
            if (notasAlumnos[i][j] < min)
                min = notasAlumnos[i][j];
    return min;
}
```

# Solución

```
void ObtenerPromedio(int notasAlumnos[ALUMNOS][NOTAS])
{
    int i, j;
    float suma;
    for (i = 0; i <= ALUMNOS - 1; i++)
    {
        suma = 0;
        for (j = 0; j <= NOTAS - 1; j++)
            suma += notasAlumnos[i][j];
        printf_s("El promedio del Alumno[%d] es: %.2f \n", i+1, suma/NOTAS);
    }
}
```

# Solución

```
C:\Users\bmuri\source\repos\ConsoleApplication1\Debug\ConsoleApplica
Ingrese las notas del alumno 1
Ingrese la nota [1]: 13
Ingrese la nota [2]: 16
Ingrese la nota [3]: 19
Ingrese la nota [4]: 11

Ingrese las notas del alumno 2
Ingrese la nota [1]: 20
Ingrese la nota [2]: 15
Ingrese la nota [3]: 16
Ingrese la nota [4]: 10

Ingrese las notas del alumno 3
Ingrese la nota [1]: 16
Ingrese la nota [2]: 6
Ingrese la nota [3]: 12
Ingrese la nota [4]: 19

La matriz es:

      [1]    [2]    [3]    [4]
Notas de Alumno[1] 13    16    19    11
Notas de Alumno[2] 20    15    16    10
Notas de Alumno[3] 16     6    12    19

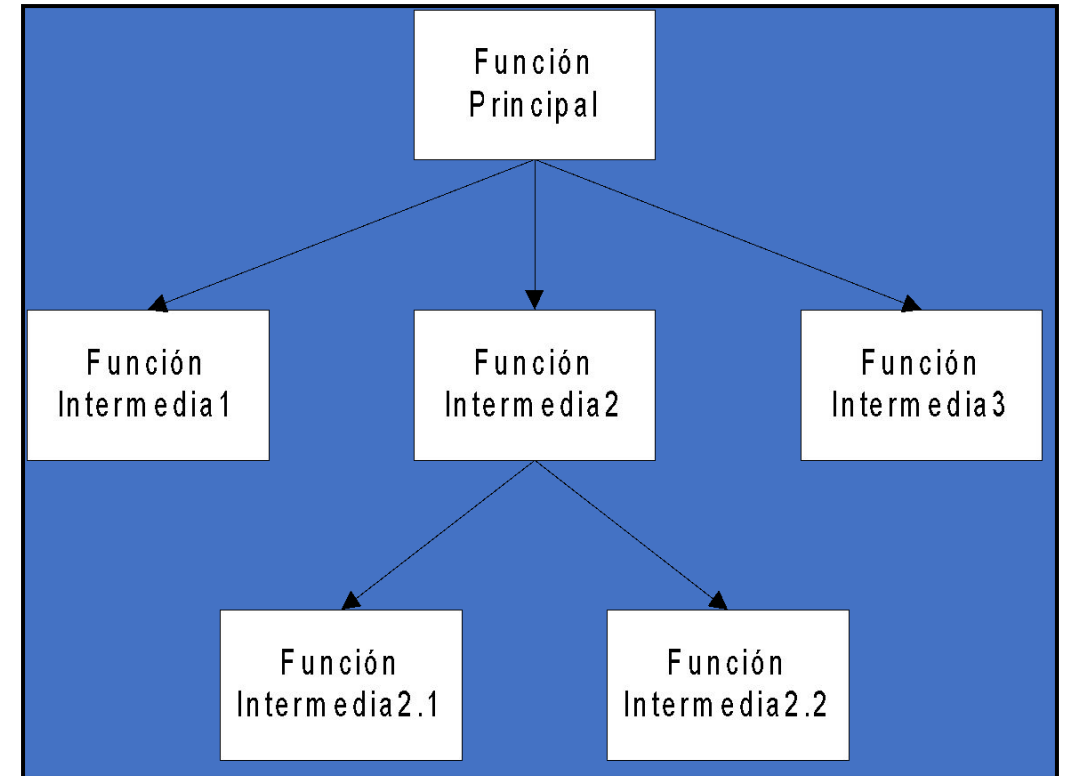
Minima nota: 6
Maxima nota: 20
El promedio del Alumno[1] es: 14.75
El promedio del Alumno[2] es: 15.25
El promedio del Alumno[3] es: 13.25
```

# Funciones



# Introducción

- Técnica del diseño descendente
- Dividir la complejidad de un problema en módulos menos complejos.
- Cada parte es más fácil de implementar (codificar) que el programa completo.
- El programador controla lo que el módulo hace.



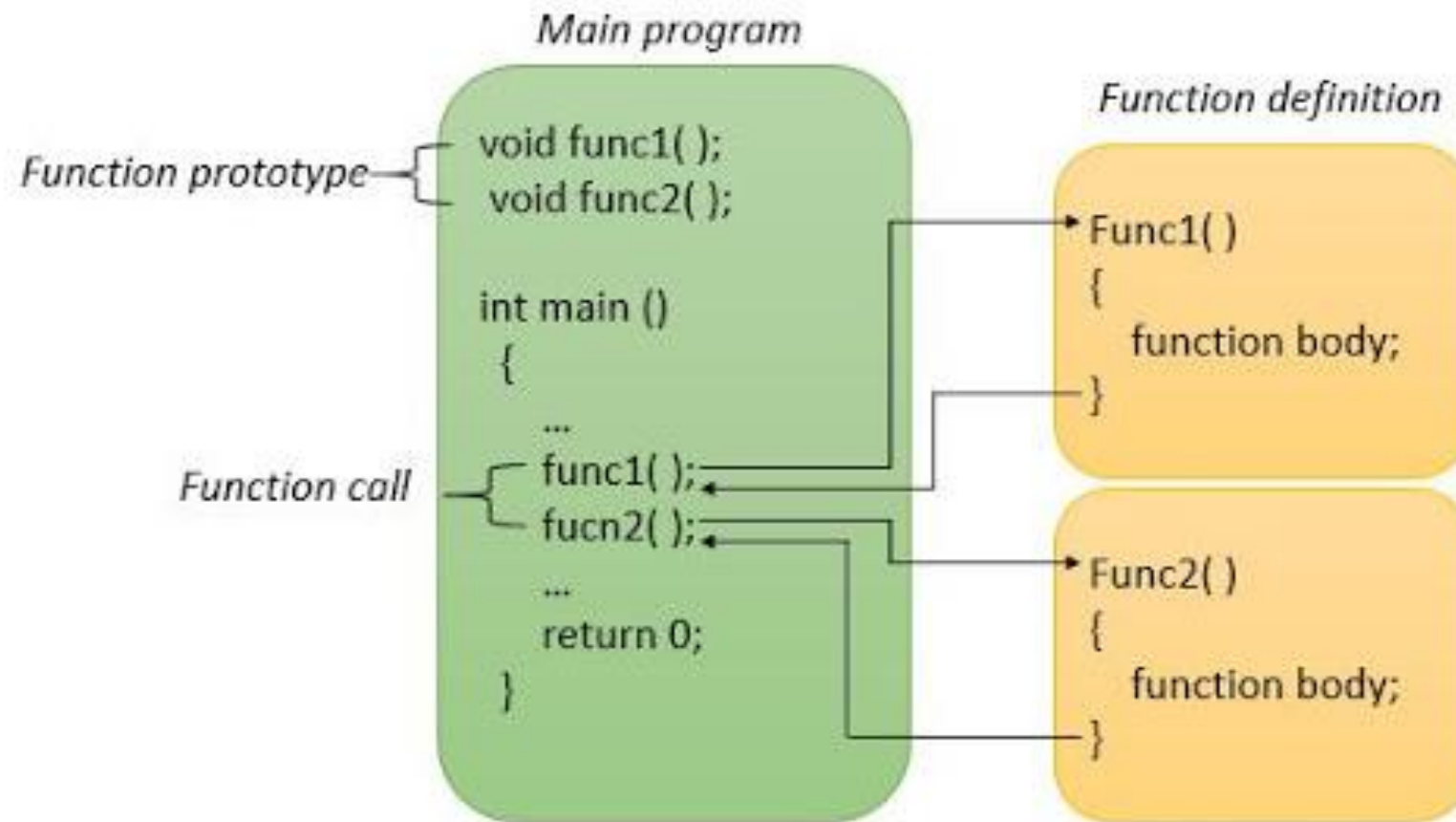
# Componentes de los programas en C++

- Los programas en C++ están compuestos por pequeños módulos definidos por el programador para realizar determinadas acciones: funciones.
- Las funciones realizan tareas específicas:
  - Operaciones matemáticas particulares
  - Manipulación de datos
- Las funciones retornan resultados calculados.
- Estas funciones se pueden ejecutar tantas veces sea necesario y desde diversos puntos del programa.

# Cómo hacer funciones propias en C++

- Paso 1: Identificar las características de la función:
  - Qué se quiere calcular: Retorno de la función
  - Qué datos necesita para efectuar el cálculo: Parámetros de la función
  - Cómo hará el cálculo
- Paso 2:
  - Escribir la declaración de la función
  - Escribir el prototipo de la función

# Cómo hacer funciones propias en C++



# Ejemplo de función en C++

- Función Factorial()

Nombre de la función

Parámetro de la función

```
long Factorial(int n)
```

Tipo de Retorno

```
{  
    long fact = 1;  
    for (int i = 1; i <= dato; i++)  
    {  
        fact = fact * i;  
    }  
    return fact;  
}
```

Resultado calculado por la función

# Funciones: Elementos

- Todas las funciones deben tener un nombre
  - Debe cumplir con las reglas de formación de los identificadores
- Las funciones pueden tener uno o varios parámetros
  - No confundir parámetros con variables auxiliares
- Las funciones pueden devolver un valor de algún determinado tipo
  - Se debe especificar si la función devuelve o no valor, de forma explícita

# Funciones: Declaración

- Formato de definición de una función:

```
Tipo_dato_retorno Nombre_Funcion(lista_parámetros)
{
    cuerpo_de_la_funcion
}
```

- El nombre de la función.
- El tipo de dato de retorno:
  - Si no se quiere devolver valor alguno, usar void.
  - Si no se indica, se asume que es tipo int.
- La lista de parámetros.

# Funciones: Uso de instrucción return

- La palabra reservada return indica el valor que la función retorna o devuelve al finalizar.
- Normalmente, es la última instrucción del cuerpo de la función.
- Cuando el compilador encuentra la palabra return, termina la ejecución de la función, ignorando las instrucciones que siguen.



# Funciones: Invocación y Parámetros

- Las funciones pueden ser usadas tanto en el programa principal (main) como en otras funciones definidas por el programador.
- Debe respetarse cantidad y tipo de parámetros que maneja la función, en todos los casos.

# Funciones: Prototipos

- Declaración anticipada al uso de las de las funciones.
- Solamente se indica el tipo de dato que devuelve, el nombre y los parámetros.
- Después se define en su totalidad.

# Usos generales de las funciones

- Uso 1: para cálculo de algún tipo
  - Se realizan operaciones (matemáticas) con sus parámetros.
- Uso 2: para determinar si se cumple o no cierta condición
  - Simular funciones boolean de VB.
- Uso 3: funciones que no retornan valor (void)
  - Hacen las veces de procedimientos de VBA.
  - Solo cuando se va a mostrar resultados.

# Llamada a Funciones: Por Valor

- Copia del parámetro se pasa a la función.
- Los cambios en la función no afectan al original.
- Se usa cuando la función no necesita modificar el parámetro. Se evitan cambios accidentales.

# Llamada a Funciones: Por Referencia

- Se pasa a la función el parámetro original.
- Los cambios en la función afectan al original.
- Requiere del uso de punteros.

# Llamada a Funciones

## Por Valor

- Copia del parámetro se pasa a la función
- Los cambios en la función no afectan al original
- Se usa cuando la función no necesita modificar el parámetro. Se evitan cambios accidentales.

## Por Referencia

- Se pasa a la función el parámetro original
- Los cambios en la función afectan al original
- Requiere del uso de punteros

# Arreglos y Funciones

- Un arreglo puede ser pasado como parámetro de una función. Para esto, se especifica el nombre del arreglo sin corchetes:

```
int myArray[24];  
myFunction( myArray, 24 );
```

- También se pasa el tamaño del arreglo.
- El arreglo se pasa con llamada por referencia. El nombre del arreglo equivale a la dirección del primer elemento.
- La función sabe donde está almacenado el arreglo.
- Una función **NO PUEDE DEVOLVER** un arreglo como resultado.

# Arreglos y Funciones

- Pasando los elementos del arreglo
  - Se pasan con llamada por valor.
  - Se pasa el nombre del elemento y su posición: `(myArray[3])` a la función
- Prototipo de función:  

```
void modifyArray( int b[], int arraySize );
```
- Los nombres de parámetros son opcionales
  - `int b[]` puede ser escrito como `int []`.
  - `int arraySize` puede ser escrito como `int`.



# Ejemplo

```
#include "stdio.h"
//Definición de constantes
const int TAMANO = 5;
//Declaración de funciones
void ModificarArreglo(int[]);
void ModificarElemento(int);
int main()
{
    int miArreglo[TAMANO] = {0,1,2,3,4};
    printf("Efectos de pasar un arreglo entero por referencia.\n");
    printf("Los valores del arreglo original son: \n");
    for (int i = 0; i < TAMANO; i++)
        printf("%3d",miArreglo[i]);
    printf("\n");
    //Llamada por Referencia
    ModificarArreglo(miArreglo);
    printf("Los valores del arreglo modificado son: \n");
    for (int i = 0; i < TAMANO; i++)
        printf("%3d", miArreglo[i]);
    printf("\n");
    //Llamada por Valor
    printf("\nEfectos de pasar el elemento de un arreglo por valor.\n");
    printf("El valor de miArreglo[3] es %d \n",miArreglo[3]);
    ModificarElemento(miArreglo[3]);
    printf("El valor de miArreglo[3] es %d \n", miArreglo[3]);
    return 0;
}
```

# Ejemplo

```
void ModificarArreglo(int miArreglo[])
{
    for (int i = 0; i < TAMANO; i++)
        miArreglo[i] = miArreglo[i] * 2;
}
void ModificarElemento(int elemento)
{
    printf("Valor en el elemento modificado es %d\n", elemento=elemento*2);
}
```

# Ejemplo

```
void ModificarArreglo(int miArreglo[])
{
    for (int i = 0; i < TAMANO; i++)
        miArreglo[i] = miArreglo[i] * 2;
}
void ModificarElemento(int elemento)
{
    printf("Valor en el elemento modificado es %d\n",
elemento=elemento*2);
}
```

Microsoft Visual Studio Debug Console

Efectos de pasar un arreglo entero por referencia.

Los valores del arreglo original son:

0 1 2 3 4

Los valores del arreglo modificado son:

0 2 4 6 8

Efectos de pasar el elemento de un arreglo por valor.

El valor de miArreglo[3] es 6

Valor en el elemento modificado es 12

El valor de miArreglo[3] es 6

# Bibliografía

- P. Deitel and H. Deitel. *C how to program : with an introduction to C++*. Pearson, Boston, 2016.
- Kernighan, B., & Ritchie, D. (1988). *The C programming language* (2nd ed.). Prentice Hall
- Stroustrup, B. (2014). *A tour of C++*. Pearson Education.
- Stroustrup, B. (2014). *Programming: Principles and Practice Using C++ (2nd ed.)*. Addison-Wesley.
- Stroustrup, B. (2018). *The C++ programming language* (4th ed.). Addison-Wesley.
- Stroustrup, B. (1994). *The design and evolution of C++*. Reading, Mass: Addison-Wesley.