

ALGORITMOS Y PROGRAMACIÓN

Clase 2

Lenguaje C# - Colecciones.

Temario

- Lenguaje C#
 - Arreglos
 - Strings
 - Funciones
 - Colecciones

Arreglos

- Un arreglo es un conjunto de **elementos homogéneos** (mismo tipo) **accedidos a través de uno o más índices.**
- La cantidad de dimensiones de un arreglo se denomina **rank**. Puede ser:
 - Unidimensional o vector (se usa un solo índice)
 - Bidimensional o matriz (se usan 2 índices)
 - Tridimensional o tensor, (se usan 3 índices)
 - N dimensional para $N \geq 4$, (se usan N índices)

Arreglos

- Arreglos de una dimensión - vectores

int [] vector1; → declaración de un vector de elementos enteros

vector1 = new int [200]; → creación del vector en memoria, con capacidad para 200 elementos enteros

int [] vector2 = **new int [100];** → declaración y creación

int [] vector3 = **new int [] {5,1,4,0};** → declaración, creación e inicialización de un vector de 4 elementos enteros.

int tam=5;

char [] vocal = **new char [tam];**

Arreglos

- Arreglos de una dimensión (vectores)

```
int [ ] vec = new int [ ] {5,1,4,0};  
int x;
```

El acceso a los elementos del vector se realiza con un índice usando el operador []

vec[0]=18 → el primer elemento está en la posición 0

```
Console.WriteLine(vec[2]);
```

x= vec.Length; → la longitud de vec es 4 pero el ultimo elemento está en la posición 3

Arreglos

- Para recorrer todos los elementos de un arreglo, se puede utilizar ***for*** o ***foreach***

```
int [ ] vec = new int [4];
```

```
for (int i=0; i< 4; i=i+1)  
    Console.WriteLine(vec[i]);
```

**Al crear el arreglo
en memoria, se
inician sus
elementos en 0.**

En este caso la variable de control del for se utiliza como índice del vector vec. El vector se recorre desde la posición 0 a la 3 para imprimir sus elementos.

Arreglos

```
int [ ] vec = new int [ 4] ;
```

- Para cargar datos en una arreglo:

```
for (int i=0; i< 4; i=i+1)
{
    Console.WriteLine("ingrese un valor entero");
    vec[i]=int.Parse(Console. ReadLine());
}
```

- Para sumar los elementos del arreglo:

```
int suma=0;
for (int i=0; i< 4; i=i+1)
{
    suma+=vec[i];
}
Console.WriteLine("Suma final" + suma);
```

Arreglos

El **foreach** reemplaza al for y se usa para iterar sobre los elementos del arreglo. No usa índice.

```
foreach (<tipoElem> <elem> in <vector>) {  
    <instrucciones>  
}
```

La variable **elem** se toma como variable de control del bloque y debe ser del mismo tipo que los elementos del arreglo.

No puede ser modificada dentro del cuerpo del foreach.

En cada repetición **elem** se asocia al elemento siguiente en forma automática (es un puntero).

Arreglos

- /*suma de los elementos del vector e impresión de los mismos, con un foreach*/

- ```
int sum=0;
int [] vec = new int[] {4,10,5};
```

```
foreach (int el in vec)
```

```
{ Console.WriteLine(el);
```

 → la variable **el** es un puntero a cada elemento del vector. Cambia en forma automática.  

```
 sum=sum + el;
```

```
}
```

```
Console.WriteLine("Suma final" + sum);
```

 → imprime 19

# Arreglos

- /\*suma de los elementos del vector e impresión de los mismos, con un for \*/

- `int sum=0;`  
`int [ ] vec = new int[ ] {4,10,5};`

```
for (int i=0; i< 3; i=i+1)
{
 Console.WriteLine(vec[i]);
 sum=sum + vec[i];
}
```

`Console.WriteLine("Suma final" + sum);` → imprime 19

# For vs Foreach para recorrer arreglos

## Diferencias entre el for y el foreach:

- foreach no usa índices; el for sí.
- El bloque del for se ejecuta para cada valor del índice en el rango especificado. El bloque del foreach se ejecuta para cada elemento de la colección.
- Con el for se puede recorrer una colección desde el principio hasta el fin o desde el fin hasta el principio en base al valor del índice. Con foreach sólo se recorre la colección de principio a fin (unidireccional).

# Arreglos multidimensionales

- Se pueden definir arreglos de dos o más dimensiones

**int [ , ] matriz = new int [3, 4];** → tiene 3 filas y 4 columnas

**int [ , , ] tensor = new int [4, 3, 8];**

- Acceso a cada elemento usando índices:

matriz [2, 1] = 5;

tensor [3, 1, 0] = 34;

# Arreglos multidimensionales

- Declaración, instanciación e inicialización

```
int [,] matriz = new int [,] { { 1, 2, 3, 4 },
 { 5, 6, 7, 8 },
 { 9, 10, 11, 12 } } ;
```

Tiene 3 filas.

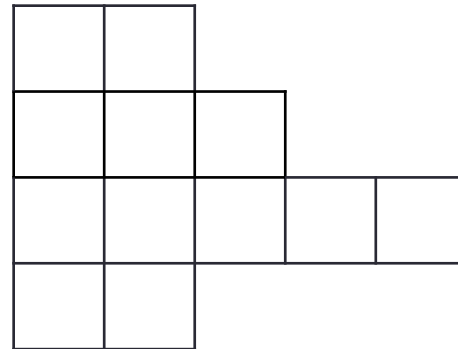
|   |    |    |    |
|---|----|----|----|
| 1 | 2  | 3  | 4  |
| 5 | 6  | 7  | 8  |
| 9 | 10 | 11 | 12 |

Y 4 columnas.

# Arreglos de arreglos

- Se pueden crear arreglos de arreglos para crear estructuras no cuadradas.

```
int [][] tabla = new int [4][];
tabla [0] = new int [2];
tabla [1] = new int [3];
tabla [2] = new int [5];
tabla [3] = new int [2];
```



**Se usan múltiples corchetes como dimensiones se necesiten. Se declara la longitud de la primer dimensión.**

# Arreglos de arreglos

- Se pueden crear arreglos de arreglos para crear estructuras no cuadradas.

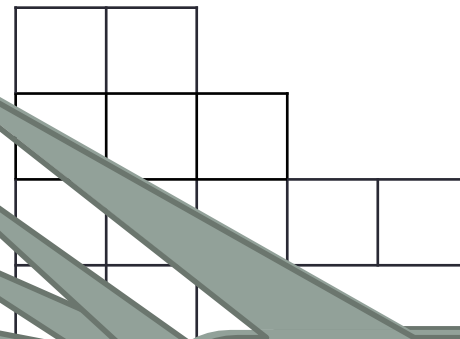
```
int [][] tabla = new int [4][];
```

```
tabla [0] = new int [2];
```

```
tabla [1] = new int [3];
```

```
tabla [2] = new int [5];
```

```
tabla [3] = new int [2];
```



Luego se declara,  
para cada fila, las  
dimensiones de  
cada subarreglo por  
separado

# Arreglos de arreglos

- Se pueden crear arreglos de arreglos para crear estructuras no cuadradas.

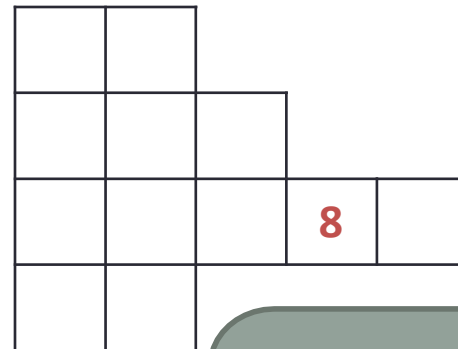
```
int [][] tabla = new int [4][];
```

```
tabla [0] = new int [2];
```

```
tabla [1] = new int [3];
```

```
tabla [2] = new int [5];
```

```
tabla [3] = new int [2];
```



```
tabla [2] [3] = 8;
```

El acceso a los elementos también es por múltiples pares de corchetes



# Puesta en común

Implemente un programa que pida al usuario que ingrese por teclado una serie de 10 números y que luego imprima aquellos números ingresados que resultan más grandes que el promedio.

Necesitamos un  
arreglo de 10  
elementos

```
double[] numeros = new double[10];
double suma = 0, prom;
for(int i = 0; i < 10; i++){
 Console.WriteLine("Ingrese un número");
 numeros[i] = double.Parse(Console.ReadLine());
 suma+= numeros[i];
}
```

```
double[] numeros = new double[10];
double suma = 0, prom;
```

Con el for leemos y  
almacenamos los  
valores, y los  
sumamos

```
for(int i = 0; i < 10; i++){
 Console.WriteLine("Ingrese un número");
 numeros[i] = double.Parse(Console.ReadLine());
 suma+= numeros[i];
}
```

```
double[] numeros = new double[10];
double suma = 0, prom;
for(int i = 0; i < 10; i++){
 Console.WriteLine("Ingrese un número");
 numeros[i] = double.Parse(Console.ReadLine());
 suma+= numeros[i];
}
prom = suma / 10;
Console.Write("Los valores mayores al promedio " +
 prom + " son: ");
```

```
double[] numeros = new double[10];
double suma = 0, prom;
for(int i = 0; i < 10; i++){
 Console.WriteLine("Ingrese un número");
 numeros[i] = double.Parse(Console.ReadLine());
 suma += numeros[i];
}
prom = suma / 10;
Console.WriteLine("Los valores mayores al promedio " + prom + " son: ");

foreach(double numero in numeros){
 if(numero > prom)
 Console.Write(numero + ",");
}
```

Con el foreach se va recorriendo todo el arreglo e imprimiendo aquellos valores que son mayores que el promedio

# Strings

- Los strings pueden verse como un arreglo de tipo char.  
    `string st = "Hola";` → cada elemento es un carácter
- Declaración y armado de strings

```
string c;
```

```
c = "Hola";
```

 → los strings van entre comillas dobles

```
c += " Mundo";
```

 → el operador +, concatena

```
c += '!';
```

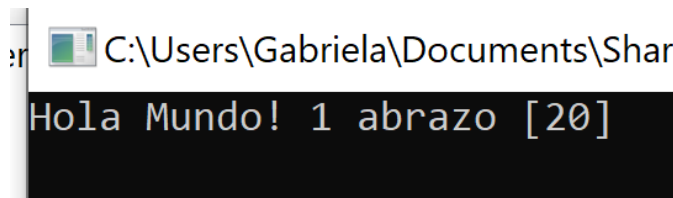
 → los caracteres van entre comillas simples

```
c += 1;
```

 → permite concatenar números, los convierte a string

```
c += " abrazo";
```

```
Console.WriteLine(c + " [" + c.Length + "]");
```

 → Length devuelve la cantidad de letras del string

# Strings

- Funciones de los strings

```
string i;
```

```
char c = 'A';
```

```
int n = 3;
```

```
i = c.ToString();
```

```
i = n.ToString();
```

Para asignarle un valor caracter o numérico al string se debe realizar una conversión explícita.

```
i = "hola mundo";
```

```
int a = i.IndexOf("mundo"); // a=5
```

```
i = i.Insert(a-1, " mi lindo "); // i="hola mi lindo mundo"
```

```
i = i.Remove(a, 3); // i="hola lindo mundo"
```

desde la posición a, elimina 3 caracteres

# Strings

- Funciones de los strings

Replace reemplaza toda  
ocurrencia del primer  
string por el segundo

```
i = i.Replace("mundo", "MunDo ");
```

```
// i="hola lindo MunDo"
```

```
string s = i.Substring(10, 4); // retorna un nuevo string tomando el contenido
de i desde la posición 10, los 4 caracteres siguientes
```

```
s = s.ToLower(); //pasa todo a minúscula
```

```
s = s.ToUpper(); //pasa todo a mayúscula
```

```
s = s.Trim(); // saca blancos del principio y del final
```

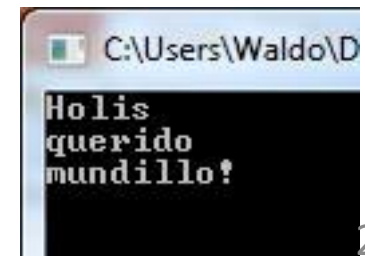


# Strings

- La función **Split** divide un string en diferentes partes buscando como separador uno o más caracteres específicos.
- Retorna un arreglo de strings, sin el separador.

```
string s = "Holis querido mundillo!";
string [] partes; // vector de strings
partes = s.Split(new char [] { ' ' });
foreach(string parte in partes)
 Console.WriteLine(parte);
```

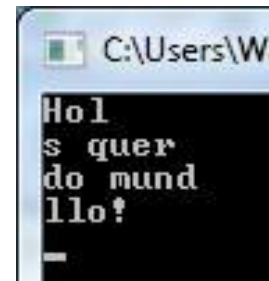
El espacio en blanco es  
usado para la división, no  
forma parte de ningún string



```
C:\Users\Waldo\D
Holis
querido
mundillo!
```

# Strings

```
string s = "Holis querido mundillo!";
string [] partes;
partes = s.Split(new char [] {'i'});
foreach(string parte in partes)
 Console.WriteLine(parte);
```



# Puesta en común Ejercicio 3 – TP 2

**Escriba un programa de aplicación que lea 2 palabras y determine si son palíndromos entre ellas. (Ej: “abbccd” y “dccbba” son palíndromos).**

- ¿Dónde almacenamos las palabras?
- ¿Qué estructura de repetición puedo usar para recorrer los strings? .....
- ¿Cómo verifico si son capicúas?
- También podríamos pensar cuándo NO lo son, para no procesar de más.

# Puesta en común Ejercicio 3 – TP 2

```
if (pal1.Length==pal2.Length)
{
 son= true;

 tam=pal1.Length;
 k=tam-1; /*ultima posición de la palabra1 y de la palabra2*/

 for(int j=0;j<tam;j++)
 { if (pal1[j] != pal2[k]) /*si encuentro una letra distinta, corto el recorrido*/
 { son=false;
 break;}
 else
 k=k-1;
 }
}

else
 son=false;
```

# Colecciones

- Una **colección** es un objeto que internamente se gestiona como un array, pero brinda distintas funcionalidades.
- Según el tipo de colección que sea:
  - algunas se acceden por clave y no por índice,
  - otras crecen de tamaño dinámicamente,
  - otras almacenan sus elementos en forma ordenada, etc..
- Aunque .NET tiene decenas de tipos de colecciones donde cada una aporta algún manejo especial para ciertos problemas, la **única colección que usaremos en este curso es el ArrayList**

# ArrayList

- Para poder hacer uso de los ArrayList se debe incluir un espacio de nombres (namespace) llamado System.Collections

```
using System;
using System.Collections;

namespace Ejercicio1
{
 class Program
 {
 public static void Main(string[] args)
 {
 ArrayList a = new ArrayList();
 }
 }
}
```

Esta cláusula nos permite utilizar todos los tipos de colecciones disponibles, entre ellos el ArrayList

# ArrayList

- El ArrayList es una colección que se redimensiona dinámicamente.
- Para poder usar un ArrayList primero hay que instanciarlo (como a los arreglos)

```
ArrayList al1 = new ArrayList();
```

Declara y crea vacío

```
ArrayList al2 = new ArrayList(5);
```

Declara y crea un ArrayList vacío para almacenar 5 elementos

```
ArrayList al3= new ArrayList(3) {12,5,"sol"};
```

Declara, crea e inicializa

```
Console.WriteLine(al3.Count); // imprime 3
```

# ArrayList

Los ArrayList admiten que se apliquen muchas operaciones sobre ellos.

Ejemplo

```
ArrayList a = new ArrayList();
double d = 4.67;
a.Add(d);
```

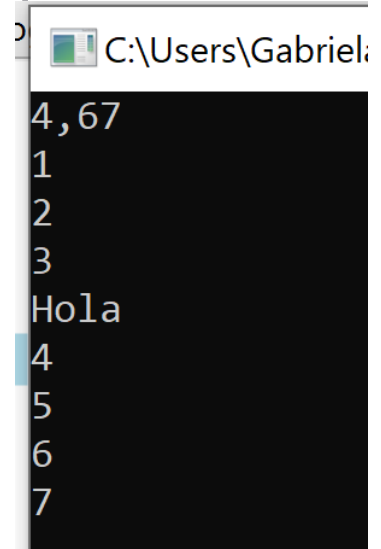
```
int [] vec = new int [] {1,2,3,4,5,6,7};
```

```
a.AddRange(vec); → agrega todos los elementos de vec
```

```
a.Insert(4, "Hola"); → desplaza elementos
```

```
for(int x=0; x < a.Count; x++)
```

```
 Console.WriteLine(a[x]); → recorrido con for
```





# ArrayList

- Los elementos de los ArrayList pueden ser de cualquier tipo (heterogéneos), por lo tanto para poder ser usados deben ser casteados al tipo correspondiente.

```
ArrayList a = new ArrayList(new int[] {1, 2.34, "5"});
```

```
int i = (int) a[0];
```

```
double d = (double) a[1];
```

```
string s = ((string) a[2]) + " peras";
```

Obviamente hay que saber el tipo de cada elemento del vector. Se aplica el cast type

# ArrayList

- Los elementos de los ArrayList también pueden ser recorridos con foreach, ***siempre que todos sus elementos sean del mismo tipo:***

```
foreach(int ele in a)
 Console.WriteLine(ele);
```

Debe ser del mismo tipo que los elementos del ArrayList (ArrayList homogéneo)

- **Atención:** Si los elementos son heterogéneos, solo se puede usar for para recorrido.

# Puesta en común Ejercicio 4 –TP 2

Haga un programa de aplicación que lea por consola una sucesión de palabras que termina con la palabra vacía. Imprima el porcentaje de palabras que comienzan con 's', la longitud de cada palabra leída y el promedio de caracteres por palabra.

**Ayuda: si `st` es una variable de tipo `string`, `st.Length` devuelve la cantidad de caracteres del `string`.**

- Si quiero que el cálculo se compute luego de haber ingresado todos los datos ¿cómo lo implemento?
- ¿En donde guardo las palabras ingresadas?
- Recordar que no se sabe cuántas palabras ingresará el usuario.
- Ayuda: `palabra != ""` o `palabra == ""`.

# Subprogramas

- Definición

```
static <tipoRetorno> <nombreSubprog>(<parámetros>)
{
 <cuerpo>
}
```

- La definición comienza con la palabra **static**.
- Si se va a definir una función se debe indicar **qué tipo de valor devuelve** en <tipoRetorno> y la sentencia return en el <cuerpo>.
- Si es un procedimiento se utiliza **void** como <tipoRetorno> y se omite el return, ya que los resultados los devuelve a través de parámetros

# Procedimientos y Funciones

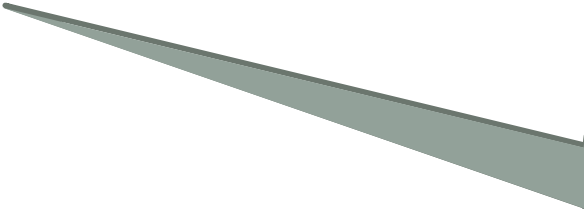
## Ejemplos

`/*procedimiento*/`

```
static void imprimir(string pal) {
 Console.WriteLine(pal);
}
```

`/*función*/`

```
static double sumar(double a, double b) {
 return a + b;
}
```



La instrucción `return` determina el valor que devuelve la función.

# Procedimientos y Funciones

## Ejemplos

```
static void imprimir(string pal) {
 Console.WriteLine(pal);
}
```

```
static double sumar(double a, double b) {
 return a + b;
 a=b;
}
```

Cualquier instrucción debajo de un return NO se ejecuta. La instrucción return "finaliza" la ejecución de cualquier función

# Procedimientos y Funciones

```
using System;

namespace Ejercicio3
{
 class Program
```

```
 {
 public static void Main(string[] args)
 {
 double sum;
 sum= sumar(3.4, 5.8);
 imprimir(sum.ToString());
 Console.ReadKey(true);
 }
 }
```

Invocación a las funciones y procedimientos

```
static void imprimir(string s)
{
 Console.WriteLine(s);
}
```

```
static double sumar (double a, double b)
{
 double res;
 res=a+b;
 return res;
}
```

Todos los subprogramas que declaremos deben estar fuera del Main. Y dentro de la sección class Program y comenzar con la palabra static.

# Repasando parámetros

Un parámetro es un medio o mecanismo de comunicación para compartir datos entre unidades de programa.

De acuerdo al lugar de uso se denominan:

- \* **actuales**: se especifican en la invocación a un subprograma
- \* **formales**: se especifican en el encabezamiento del subprograma

```
namespace Ejercicio3
{
 class Program
 {
 public static void Main(string[] args)
 {
 double sum;
 sum = sumar(3.4, 5.8) //parámetros actuales
 Console.ReadKey(true);
 }

 static double sumar (double a, double b) //p.formales
 {
 return a + b;
 }
 }
}
```

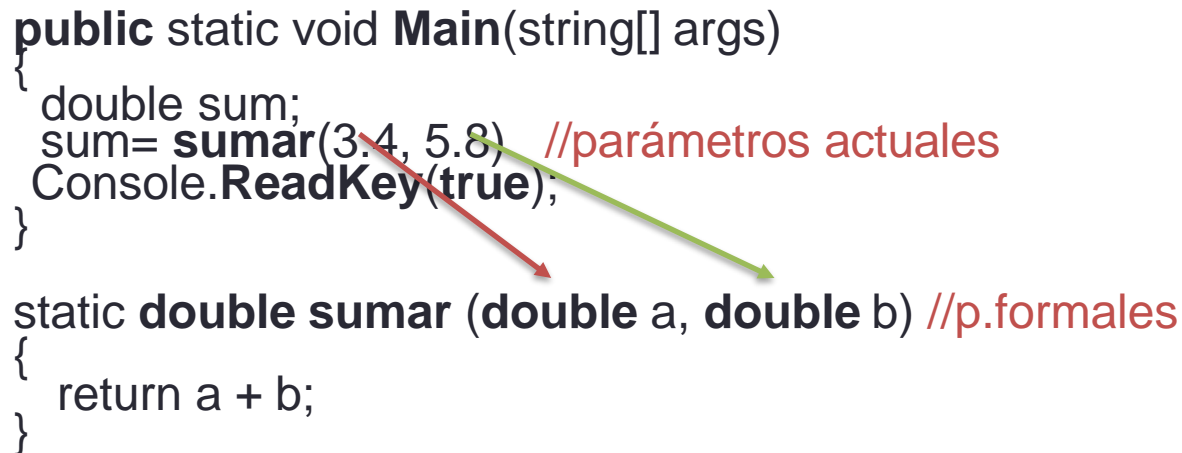


Cada lenguaje de programación presenta reglas propias respecto de la forma de pasar los parámetros.

En caso de haber **correspondencia posicional** se asocia el primer parámetro actual con el primero formal, y así sucesivamente. En ese caso deben coincidir en posición, tipo y cantidad de parámetros.

```
public static void Main(string[] args)
{
 double sum;
 sum= sumar(3.4, 5.8) //parámetros actuales
 Console.ReadKey(true);
}

static double sumar (double a, double b) //p.formales
{
 return a + b;
}
```



# Tipos de Parámetros

Los parámetros se clasifican según su uso en:

- **De entrada**: llevan datos a la unidad invocada, pero no retornan resultados a quien la invoca
- **De salida**: no llevan datos a la unidad invocada, pero retornan resultados a quien la invoca.
- **De entrada-salida**: llevan datos a la unidad invocada y retornan resultados a quien la invoca.

De acuerdo al uso de los parámetros se pueden definir distintos tipos de **pasaje de parámetros**.

Por **copia o valor**: si el parámetro formal es de entrada.

Por **referencia**: si el parámetro formal es de salida o entrada salida

# Pasaje de Parámetros

Por **copia o valor**: el parámetro formal copia el valor del parámetro actual. Cualquier modificación sobre el mismo no modifica al dato original.

Por **referencia**: el parámetro formal toma la dirección de memoria del parámetro actual, por lo cual cualquier modificación que se realice sobre el parámetro formal, en realidad se realiza sobre la variable original.

Por defecto en C# los parámetros se pasan por copia. Para que sean pasados por referencia debe especificarse con la palabra clave **ref** (tanto en la invocación como en el encabezado).

# Pasaje de Parámetros

```
using System;
```

```
namespace proyecto2
```

```
{
```

```
class Program
```

```
{
```

```
public static void Main(string[] args)
```

```
{
```

```
int res, nro1, nro2;
```

```
Console.WriteLine("ingrese primer valor:");
```

```
nro1=int.Parse(Console.ReadLine());
```

```
Console.WriteLine("ingrese segundo valor:");
```

```
nro2=int.Parse(Console.ReadLine());
```

```
res=suma(nro1,nro2);
```

-----> se usan parámetros de entrada

```
Console.WriteLine("La suma de los valores es {0}",res);
```

```
Console.Write("Press any key to continue . . . ");
```

```
Console.ReadKey(true);
```

```
}
```

```
static int suma (int val1, int val2)
```

-----> pasaje por copia o valor

```
{
```

```
return val1 + val2;
```

```
}
```

```
}
```

```
}
```

Dados 2  
números  
enteros  
obtener e  
imprimir la  
suma.

# Pasaje de Parámetros

```
namespace proyecto2
```

```
{
 class Program
 {
 public static void Main(string[] args)
 {
 double salario;
 int porcaumento;
 Console.WriteLine("ingrese el salario actual del empleado:");
 salario=double.Parse(Console.ReadLine());
 Console.WriteLine("ingrese porcentaje de aumento:");
 porcaumento=int.Parse(Console.ReadLine());

 modificaSueldo(ref salario,porcaumento);

 Console.WriteLine("El sueldo actualizado es {0}",salario);
 Console.ReadKey(true);
 }
 }
}
```

```
static void modificaSueldo(ref double sueldo, int porc)
```

```
{
 sueldo= sueldo + porc*suelo/100;
}
}
```

Dado el sueldo de un empleado obtener el sueldo actualizado al aplicarle un porcentaje de aumento.

-----> el salario es un parámetro de entrada/salida

-----> la variable sueldo se pasa por referencia

# Puesta en común Ejercicio 6 - TP 2

**Defina un subprograma que reciba una palabra y retorne en un vector la cantidad de cada una de las vocales que contiene.**

- ¿Qué tipo de arreglo vamos a usar?
- ¿De que tipo van a ser sus elementos?
- Si el vector es de contadores, ¿cómo debemos inicializarlo?
- ¿Usamos una función o un procedimiento?
- ¿Qué tipo de parámetros debemos usar?
- De acuerdo a cómo usamos los parámetros, cuál será el pasaje a utilizar?

```
string pal;
int k;
int [] cantvoc=new int[5];
char [] nom=new char[5]{'a','e','i','o','u'};
for(k=0;k<cantvoc.Length;k++)
 cantvoc[k]=0;
```

```
Console.WriteLine("ingrese una palabra");
pal=Console.ReadLine();
```

```
contar(pal, ref cantvoc); /*invocación*/
```

```
k=0;
foreach (int el in cantvoc)
{ Console.WriteLine("cantidad de letras " + nom[k] + " es " + el);
 k=k+1;
}
Console.Write("Press any key to continue . . . ");
Console.ReadKey(true);
}
```

```
/*definición del procedimiento que cuenta las vocales*/
public static void contar(string p, ref int [] vec)
{
 for(int i=0;i<p.Length;i++)
 { char c;
 c=p[i]; /*se toma cada letra de la palabra*/
 switch (c)
 {
 case 'a': vec[0]=vec[0]+1;break;
 case 'e': vec[1]=vec[1]+1;break;
 case 'i': vec[2]=vec[2]+1;break;
 case 'o': vec[3]=vec[3]+1;break;
 case 'u': vec[4]=vec[4]+1;break;
 }
 }
}
```



# Muchas gracias

