



# Fundamentos de programación

## Capítulo 1: Paradigmas de la programación

Profesor: Carlos Diaz

cdiazd@uni.edu.pe

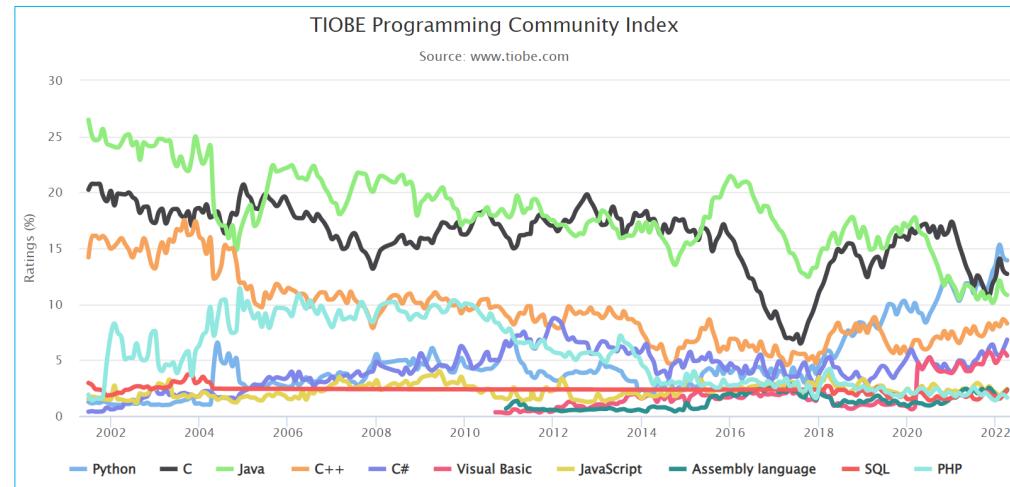
## Contenido

- Índice Tiobe Abril 2022
- Paradigmas de la programación
- Programación imperativa
  - Programación no estructurada
  - Programación estructurada
  - Programación procedural
  - Programación modular
- Programación declarativa
  - Programación lógica
  - Programación funcional
- Programación orientada a objetos

## Índice Tiobe Abril 2022

Apr 2022	Apr 2021	Change	Programming Language	Ratings	Change
1	3	▲	Python	13.92%	+2.88%
2	1	▼	C	12.71%	-1.61%
3	2	▼	Java	10.82%	-0.41%
4	4		C++	8.28%	+1.14%
5	5		C#	6.82%	+1.91%
6	6		Visual Basic	5.40%	+0.85%
7	7		JavaScript	2.41%	-0.03%
8	8		Assembly language	2.35%	+0.03%
9	10	▲	SQL	2.28%	+0.45%
10	9	▼	PHP	1.64%	-0.19%

## Índice Tiobe Abril 2022



# Paradigmas de la programación

- Un paradigma de programación es un estilo de desarrollo de programas. Es decir, un modelo para resolver problemas computacionales.



## Programación no estructurada

- La programación no estructurada es el paradigma de programación históricamente más antiguo.
- En la programación no estructurada, el código se escribe como un bloque completo único, secuencial.
- Es más difícil hacer cambios en el programa y a medida que crece se hace más difícil de leer.
- Se caracteriza por el uso de flujo de control no estructurado **GOTO** para saltar a cualquier línea de código o **GOSUB** para saltar a una **subrutina**.
- Los lenguajes no estructurados son Assembler y las primeras versiones de BASIC y Fortran.
- En el siguiente link tenemos un compilador BASIC online:  
<https://www.quitebasic.com/>

❖ Una **subrutina** ejecuta un conjunto de instrucciones sin devolver un valor.  
❖ Una **función** ejecuta un conjunto de instrucciones y devuelve un valor.

BASIC Program

```
10 LET t1=0
20 PRINT (t1)
30 LET t2=1
40 PRINT (t2)
50 LET t3=t1+t2
60 PRINT (t3)
70 LET t1=t2
80 LET t2=t3
90 LET n=n+1
100 IF n<15 THEN GOTO 50
110 PRINT ("FIN")
```

Output

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
FIN
```

## Programación imperativa

- La programación imperativa (del latín imperare = ordenar) es el paradigma de programación más antiguo (principio de los años 50).
- De acuerdo con este paradigma, un programa consiste en una secuencia claramente definida de instrucciones para la computadora.
- El código fuente de los lenguajes imperativos encadena instrucciones una detrás de otra que determinan lo que debe hacer la computadora en cada momento para alcanzar un resultado deseado. Los valores utilizados en las variables se modifican durante la ejecución del programa.
- Los lenguajes de programación imperativa más conocidos actualmente son: Java, C, C#, C++, Assembly, BASIC, Python, etc.
- La programación imperativa se clasifica a su vez en distintos estilos de programación: el no estructurado, el estructurado, el procedural y el modular.

## Programación no estructurada

- Programa que calcula la hipotenusa de un triángulo rectángulo utilizando GOSUB.

BASIC Program

```
10 INPUT "Ingrese el cateto a: "; a
20 INPUT "Ingrese el cateto b: "; b
30 IF a<0 OR b<0 THEN GOSUB 600
40 GOSUB 500
50 LET a=7
60 LET b=24
70 GOSUB 500
80 PRINT "Fin"
90 END
500 REM Calculo de hipotenusa
510 PRINT "Triangulo de cateto: "; a
520 PRINT "Triangulo de cateto: "; b
530 LET c = SQR (a*a + b*b)
540 PRINT "La hipotenusa es: "; c
550 RETURN
600 REM Subrutina de error
610 PRINT "Hubo un error"
620 GOTO 50
```

Output

```
Triangulo de cateto: 8
Triangulo de cateto: 12
La hipotenusa es: 14.42205101855956
Triangulo de cateto: 7
Triangulo de cateto: 24
La hipotenusa es: 25
Fin
```

Clear!

## Programación estructurada

- La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras de control (secuencial, condicional, iterativa).
- Este tipo de programación surgió a finales de los años 70 y se basó en el **teorema del programa estructurado**, propuesto por Böhm y Jacopini (1966), que demuestra que todo programa puede escribirse utilizando únicamente tres estructuras de control:

  - Estructura secuencial. Está formada por una secuencia de llamadas a instrucciones del lenguaje o subrutinas del programador.
  - Estructura condicional (if y switch). Es aquella que ejecuta una estructura si se cumple una condición booleana .
  - Estructura iterativa con condición (for y while). Es aquella que ejecuta una estructura una y otra vez si se cumple una condición booleana .

- Queda descartado el uso de GOTO.
- Los lenguajes estructurados más utilizados son Java, C, C#, C++, Python.
- En el siguiente link tenemos un compilador C++ online:  
<https://paiza.io/>



```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     int t1,t2,t3;
5     t1=0;
6     t2=1;
7     cout<<t1;
8     cout<<"- "<<t2;
9     for(int i=0;i<15;i++)
10    {
11        t3=t1+t2;
12        cout<<"- "<<t3;
13        t1=t2;
14        t2=t3;
15    }
16    return 0;
17 }
18 
```

Ejecutar (Ctrl-Enter) |

Salida Entrada Comments 0

0-1-2-3-5-8-13-21-34-55-89-144-233-377-610-987

## Programación procedural

- La programación procedural divide el código en partes más pequeñas y manejables llamadas funciones y subrutinas.
- De esta manera se consigue que el código sea más claro y que no sean necesarias las repeticiones de código gracias a las llamadas a las funciones y subrutinas.
- En el caso de pequeños programas este tipo de organización es lo suficientemente eficientes.
- Los lenguajes procedimentales más utilizados son Java, C, C#, C++, Python.
- En el siguiente link tenemos un compilador C++ online:  
<https://paiza.io/>



```
1 #include <iostream>
2 using namespace std;
3 int f(int t1,int t2)
4 {
5     return t1+t2;
6 }
7 int main(){
8     int t1,t2,t3;
9     t1=0;
10    t2=1;
11    cout<<t1;
12    cout<<"- "<<t2;
13    for (int i=0;i<15;i++)
14    {
15        t3=f(t1,t2);
16        cout<<"- "<<t3;
17        t1=t2;
18        t2=t3;
19    }
20    return 0;
21 }
```

Ejecutar (Ctrl-Enter) |

Salida Entrada Comments 0

0-1-2-3-5-8-13-21-34-55-89-144-233-377-610-987

## Programación modular

- La programación modular es muy similar al enfoque procedural, o más bien lo adapta a los requerimientos de proyectos de software mayores y más amplios.
- En este sentido, el código fuente se divide específicamente en bloques parciales lógicos independientes los unos de los otros para proporcionar más transparencia y facilitar el proceso de depuración de errores.
- Los bloques parciales individuales, denominados módulos, se pueden probar por separado antes de vincularlos posteriormente a una aplicación conjunta.

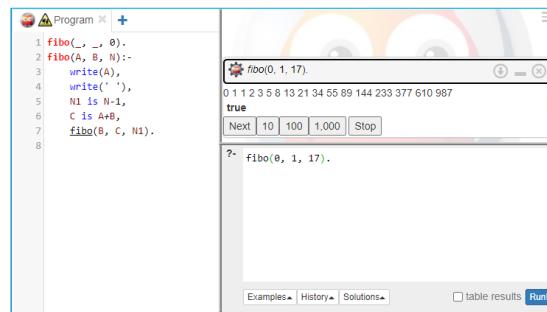


## Programación declarativa

- La programación declarativa es un paradigma de programación en el que el programador define lo que debe lograr el programa sin definir cómo debe implementarse. En otras palabras, el enfoque se centra en lo que debe lograrse en lugar de instruir cómo lograrlo.
- Este paradigma a su vez se divide en dos:
  - Programación Lógica: Prolog, etc.
  - Programación funcional: Lisp, Haskell, etc.

# Programación lógica

- La programación lógica, se basa en la lógica matemática.
- En lugar de una sucesión de instrucciones, un software programado según este principio contiene un conjunto de principios que se pueden entender como una recopilación de hechos y suposiciones.
- En el siguiente link tenemos un compilador Prolog online:  
<https://swish.swi-prolog.org/>



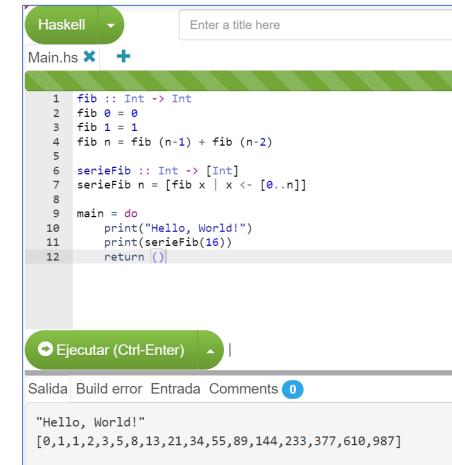
The screenshot shows a Prolog query window with the code for calculating a Fibonacci sequence. The code defines a predicate `fibo(A, B, N)` that prints the first `N` numbers in the sequence. It uses backtracking to find the next number in the sequence. The output window shows the sequence from 0 to 17. Below the code and output, there are tabs for Examples, History, Solutions, and Run.

```
1 fibo(_, _, 0).
2 fibo(A, B, N) :- !,
3   write(A),
4   writeln(' '),
5   N1 is N-1,
6   C is A+B,
7   fibo(B, C, N1).  
8  
?- fibo(0, 1, 17).
```

Output:  
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987  
true  
Next | 10 | 100 | 1.000 | Stop

# Programación Funcional

- La programación funcional se centra en las funciones.
- En un programa funcional, todos los elementos pueden entenderse como funciones y el código puede ejecutarse mediante llamadas de función secuenciales.
- En el siguiente link tenemos un compilador Haskell online:  
<https://paiza.io/>



The screenshot shows a Haskell code editor with a script named Main.hs containing code to calculate a Fibonacci sequence. The code defines a function `fib` that takes an integer and returns the `n`-th Fibonacci number. It also defines a function `serieFib` that generates a list of the first `n` Fibonacci numbers. The output window shows the result of running the code, which is "Hello, World!" followed by the sequence [0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987].

```
Haskell -> Enter a title here  
Main.hs x +  
1 fib :: Int -> Int
2 fib 0 = 0
3 fib 1 = 1
4 fib n = fib (n-1) + fib (n-2)
5
6 serieFib :: Int -> [Int]
7 serieFib n = [fib x | x <- [0..n]]
8
9 min = do
10   print("Hello, World!")
11   print(serieFib(16))
12   return ()  
  
Ejecutar (Ctrl-Enter) |  
Salida Build error Entrada Comments 0  
"Hello, World!"  
[0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987]
```

# Programación orientada a objetos

- La programación orientada a objetos se desarrolló para tratar de paliar diversas limitaciones que se encontraban en anteriores enfoques de programación.
- Cuando los programas se vuelven más grandes, la lista de instrucciones aumenta considerablemente, de modo tal que el programador tiene muchas dificultades para controlar ese gran número de instrucciones.
- Para resolver este problema los programas se descompusieron en unidades más pequeñas que adoptaron el nombre de funciones o subrutinas (paradigma procedimental)
- De este modo un programa orientado a procedimientos se divide en funciones, de modo que cada función tiene un propósito bien definido y resuelve una tarea concreta, y se diseña una interfaz claramente definida (el prototipo o cabecera de la función) para su comunicación con otras funciones.
- Con el paso de los años, la idea de romper un programa en funciones fue evolucionando y se llegó al agrupamiento de las funciones en otras unidades más grandes llamadas **módulos** (normalmente, en el caso de C++, denominadas archivos o ficheros); sin embargo, el principio seguía siendo el mismo: agrupar componentes que ejecutan listas de instrucciones (sentencias).
- Esta característica hace que a medida que los programas se hacen más grandes y complejos, el paradigma estructurado comienza a dar señales de debilidad y resulta muy difícil terminar los programas de un modo eficiente.

# Programación orientada a objetos

- Otro problema importante de la programación estructurada reside en el hecho de que la disposición separada de datos y funciones no se corresponden con los modelos de las cosas del mundo real.
- En el mundo físico se trata con objetos físicos tales como personas, autos o aviones.
- Estos objetos no son como los datos ni como las funciones. Los objetos complejos o no del mundo real tienen atributos y comportamiento.
- Los **atributos** o características de los objetos son, por ejemplo: en las personas, su edad, su profesión, su domicilio, etc.; en un auto, la potencia, el número de matrícula, el precio, número de puertas, etc.; en una casa, la superficie, el precio, el año de construcción, la dirección, etcétera. En realidad, los atributos del mundo real tienen su equivalente en los datos de un programa; toman un valor específico, tal como 200 metros cuadrados, 20.000 dólares, cinco puertas, etc.
- El **comportamiento** son las acciones que ejecutan los objetos del mundo real como respuesta a un determinado estímulo. Si usted pisa los frenos en un auto, el coche (carro) se detiene; si acelera, el auto aumenta su velocidad, etc. El comportamiento, en esencia, es como una función: se llama a una función para hacer algo (visualizar la nómina de los empleados de una empresa).
- Por estas razones, ni los datos ni las funciones, por sí mismas, modelan los objetos del mundo real de un modo eficiente.

# Programación orientada a objetos

- La programación orientada a objetos es el paradigma de programación más utilizado del siglo XXI, trae un nuevo enfoque a los retos que se plantean en la programación estructurada cuando los problemas a resolver son complejos.
- En este paradigma se construyen modelos de objetos que representan elementos (objetos) del problema a resolver, que tienen características y funciones. Permite separar los diferentes componentes de un programa, simplificando así su creación, depuración y posteriores mejoras. La programación orientada a objetos disminuye los errores y promociona la reutilización del código. Es una manera especial de programar, que se acerca de alguna manera a cómo expresaríamos las cosas en la vida real.
- La programación orientada a objetos se sirve de diferentes conceptos como:
  - Abstracción de datos
  - Encapsulación
  - Eventos
  - Modularidad
  - Herencia
  - Polimorfismo

FIN

FACULTAD DE  
CIENCIAS

ESCUELA  
PROFESIONAL DE  
**CIENCIA DE LA COMPUTACIÓN**

## Contenido

- Concepto de recursividad
- Requisito fundamental de la recursividad
- Ejercicios

# Fundamentos de programación

## Clase 02: Recursividad

Profesor: Carlos Diaz

cdiazd@uni.edu.pe

## Concepto de recursividad

- La **recursividad** (recursión) es la propiedad que posee una función de permitir que dicha función pueda llamarse directamente así misma, o bien indirectamente a través de otra función.
- Se puede utilizar la recursividad como una alternativa a la iteración.
- La recursión es una herramienta poderosa e importante en la resolución de problemas y en la programación.
- Una solución recursiva es normalmente menos eficiente en términos de tiempo de computadora que una solución iterativa; sin embargo, en muchas circunstancias el uso de la recursión permite a los programadores especificar soluciones naturales, sencillas, que serían, en caso contrario, difíciles de resolver.

## Ejemplo 1

- Sea  $x$  un número real y  $n$  un número natural, calcule  $x^n$  usando:

- Una iteración
- Una función recursiva

Solución a:  $x^n = x \cdot x \cdot x \cdots x \cdot x$  (n veces)

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     double x, p = 1;
6     int n;
7     cin >> x >> n;
8     for (int i = 0; i < n; i++)
9         p = p * x;
10    cout << "La potencia es " << p << endl;
11    return 0;
12 }
```

## Ejemplo 1 (continuación)

Solución b:  $x^n = x \cdot x^{n-1}$  (si  $n=0$  entonces  $x^0=1$ )

```
1 #include <iostream>
2 using namespace std;
3 double potencia(double x, int n)
4 {
5     if (n == 0)
6         return 1;
7     else
8         return x * potencia(x, n - 1);
9 }
10 int main()
11 {
12     double x, p;
13     int n;
14     cin >> x >> n;
15     p = potencia(x, n);
16     cout << "La potencia es " << p << endl;
17     return 0;
18 }
```

## Ejemplo 2

- Calcule recursivamente el factorial de un número  $n$ .

```
1 #include <iostream>
2 using namespace std;
3 double factorial(int n)
4 {
5     if (n==0)
6         return 1;
7     else
8         return n*factorial(n-1);
9 }
10 int main()
11 {
12     double f;
13     int n;
14     cin>>n;
15     f=factorial(n);
16     cout<<"El factorial es "<<f<<endl;
17     return 0;
18 }
```

## Requisito fundamental de la recursividad

- Un requisito para que un algoritmo recursivo sea correcto es que no genere una secuencia infinita de llamadas sobre sí mismo. En consecuencia, la definición recursiva debe incluir un componente base (**caso base** o **condición de salida**) en el que  $f(n)$  se defina directamente (es decir, no recursivamente) para uno o más valores de  $n$ .
- Toda función recursiva usa una [pila de llamada](#).
- Cada vez que se llama a una función recursiva todos los valores de los parámetros formales y variables locales son almacenados en una pila.
- Cuando termina de ejecutarse una función recursiva se retorna al nivel inmediatamente anterior, justo en el punto del programa que produjo la llamada.
- En la pila se recuperan los valores tanto de los parámetros como de las de variables locales, y se continúa con la ejecución de la siguiente instrucción a la llamada recursiva.

## Ejercicio 1

- Considere  $A$  como un arreglo de enteros. Usando recursividad:
  - Calcular el mayor elemento del arreglo
  - Calcular la suma de los elementos del arreglo

## Ejercicio 2

- Determine recursivamente si una palabra es palíndroma.

Ejemplo de palabras palíndromas:

- Rotor
- Somos
- Reconocer

- Opcional: Modifique el programa para que acepte frases palíndromas.

Ejemplo de palabras palíndromas:

- Anita lava la tina
- Somos o no somos
- Yo hago yoga hoy

## Ejercicio 3

- Convertir recursivamente un número escrito en sistema decimal al sistema binario.

## Ejercicio 4

- El algoritmo de Euclides es un conocido algoritmo para calcular el máximo común divisor de dos números.
- Se basa en la siguiente propiedad:  $\text{mcd}(a,b)=\text{mcd}(a-b,b)$ .
- Así, si se va sustrayendo el número menor del número mayor, cada vez los pares de números que quedan se van haciendo más pequeños hasta que uno de los números es 0, y  $\text{mcd}(a,0)=a$ .
- Ejemplo:  
 $\text{mcd}(105,70)=\text{mcd}(105-70,70)=\text{mcd}(35,70)=\text{mcd}(35,70-35)=\text{mcd}(35,35)=\text{mcd}(35,35-35)=\text{mcd}(35,0)=35$
- Calcular recursivamente el máximo común divisor de dos números utilizando el algoritmo de Euclides.

## Ejercicio 5

- La función de Ackerman se define en forma recursiva para enteros no negativos de la siguiente manera:

$$A(m, n) = \begin{cases} n + 1, & \text{si } m = 0 \\ A(m - 1, 1), & \text{si } m > 0 \text{ y } n = 0 \\ A(m - 1, A(m, n - 1)), & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

- Implemente la función recursiva de Ackerman.

## Ejercicio 6

- Escriba un programa que tenga una función recursiva en el cual se ingrese por teclado un número  $n$  y calcule el valor de la siguiente fracción continua:

$$2 - \cfrac{2}{2 - \cfrac{3}{3 - \cfrac{4}{4 - \cfrac{5}{\dots}}}}$$

- Para  $n=4$  debe ser:

$$a_1 - \cfrac{2}{a_2 - \cfrac{3}{a_3 - \cfrac{4}{a_4}}} = 2 - \cfrac{2}{2 - \cfrac{3}{3 - \cfrac{4}{4}}}$$

## Ejercicio 7

- Calcular recursivamente para  $n$ :

$$x = \cfrac{4}{1 + \cfrac{1}{3 + \cfrac{4}{5 + \cfrac{9}{7 + \cfrac{16}{9 + \cfrac{25}{11 + \cfrac{36}{\ddots}}}}}}}$$

- Si  $n=7$ , la fracción sería:  $x = \cfrac{4}{1 + \cfrac{1}{3 + \cfrac{4}{5 + \cfrac{9}{7}}}}$



# FIN

## Fundamentos de programación

### Clase 03: Búsqueda

Profesor: Carlos Diaz

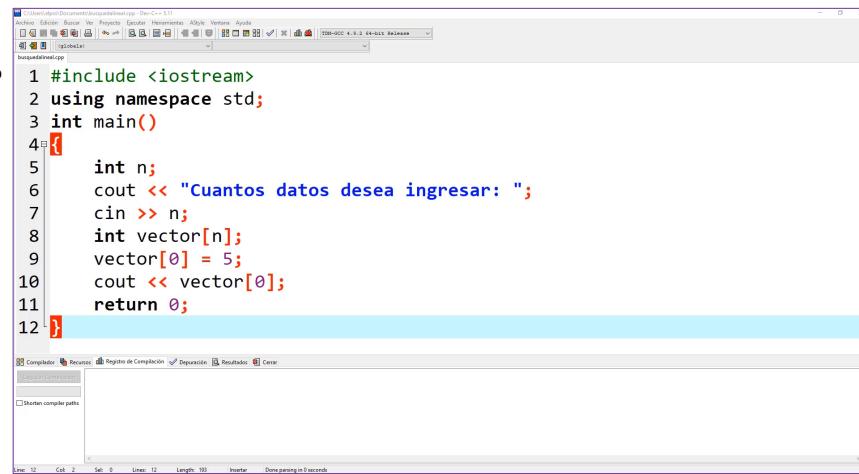
cdiazd@uni.edu.pe

## Contenido

- Búsqueda lineal o secuencial
- Búsqueda binaria

## Nota importante

- Según el estándar C++ las matrices no pueden ser de tamaño variable, es decir, no pueden definirse durante la ejecución ni definir su tamaño con variables.
- El tamaño debe ser constante en tiempo de compilación.



A screenshot of a Microsoft Visual Studio IDE window titled "busquedasec.cpp". The code editor contains the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int n;
6     cout << "Cuantos datos desea ingresar: ";
7     cin >> n;
8     int vector[n];
9     vector[0] = 5;
10    cout << vector[0];
11    return 0;
12 }
```

The code defines a function that reads an integer from the user, creates an array of that size, and then prints the first element of the array. This is a common example used to illustrate that arrays must have a constant size at compile time.

## Nota importante

- Sin embargo algunos Entornos de Desarrollo Integrado (IDE) como el Dev C++ si ejecutan este código ilegal, prohibido, clandestino, ilícito.

The screenshot shows the Dev-C++ IDE interface. In the main editor window, there is a syntax error highlighted in red at line 4, column 1, indicating a brace matching issue. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int n=10;
6     int vector[n];
7     vector[0] = 5;
8     cout << vector[0];
9     return 0;
10 }
```

The status bar at the bottom indicates "Done parsing in 0 seconds".

- En el video se muestra el IDE de Visual Studio que sigue las directivas del estándar y detecta el error.
- La solución en este caso es utilizar matrices dinámicas o la librería <vector>.

The screenshot shows the Microsoft Visual Studio IDE interface. A syntax error is highlighted in red at line 4, column 1, in the code editor. The code is identical to the one in the Dev-C++ screenshot:

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int n;
6     cout << "Cuantos datos desea ingresar: ";
7     cin >> n;
8     int vector[n];
9     vector[0] = 5;
10    cout << vector[0];
11    return 0;
12 }
```

The status bar at the bottom indicates "Uso: 100 | Código: 2 | TABLAZON: 0 | OLE".

## Nota importante

- En este caso, la solución es declarar n como una constante

const int n = 10;

The screenshot shows the Microsoft Visual Studio IDE interface with the corrected code. A constant integer 'n' is declared at the top of the main function. The code is now valid:

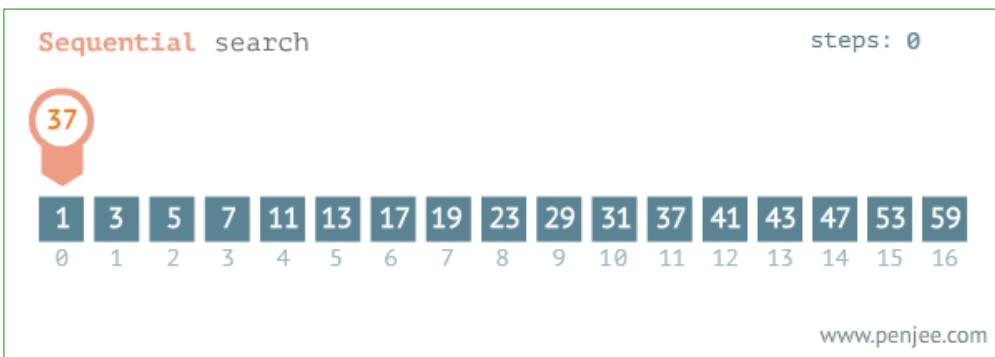
```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     const int n;
6     cout << "Cuantos datos desea ingresar: ";
7     cin >> n;
8     int vector[n];
9     vector[0] = 5;
10    cout << vector[0];
11    return 0;
12 }
```

The status bar at the bottom indicates "Uso: 100 | Código: 2 | TABLAZON: 0 | OLE".

## Búsqueda lineal o secuencial

- En una búsqueda secuencial, los elementos de una lista se exploran en secuencia, uno después de otro.
- El elemento que se busca se llama **clave** de búsqueda.
- El algoritmo de búsqueda secuencial compara cada elemento del array con la clave de búsqueda.
- Dado que el array no está en un orden prefijado, es probable que el elemento a buscar pueda ser el primer elemento, el último elemento o cualquier otro.
- De promedio, al menos el programa tendrá que comparar la clave de búsqueda con la mitad de los elementos del array.
- El método de búsqueda lineal funcionará bien con arrays pequeños o no ordenados.

## Búsqueda lineal o secuencial



## Ejercicio 1

- Lea datos desde un vector y realice la búsqueda lineal o secuencial.
  - Para datos numéricos
  - Para cadenas

## Búsqueda binaria

- La búsqueda secuencial se aplica a cualquier lista.
- Si la lista está ordenada, la búsqueda binaria proporciona una técnica de búsqueda mejorada.
- Localizar una palabra en un diccionario es un ejemplo típico de búsqueda binaria. Dada la palabra, se abre el libro cerca del principio, del centro o del final dependiendo de la primera letra de la palabra que busca. Se puede tener suerte y acertar con la página correcta; pero, normalmente, no será así y se mueve el lector a la página anterior o posterior del libro.
- Una idea similar se aplica en la búsqueda en una lista ordenada. Se sitúa el índice de búsqueda en el centro de la lista y se comprueba si nuestra clave coincide con el valor del elemento central.
- Si no se encuentra el valor de la clave, se sitúa en la mitad inferior o superior del elemento central de la lista.

## Búsqueda binaria

- Sea una lista ascendente en un array delimitado por índices **bajo** y **alto**, los pasos a seguir son:
  - Calcular el índice del punto central del array:
$$\text{central} = (\text{bajo} + \text{alto})/2$$
 (división entera)
  - Comparar el valor de este elemento **central** con la **clave**:

$\text{clave} = a[\text{central}]$  ---  bajo central alto	$\text{clave} < a[\text{central}]$  ---  bajo central alto	$\text{clave} > a[\text{central}]$  ---  bajo central alto
Clave encontrada	Búsqueda lista inferior	Búsqueda lista superior
- Si **clave < a[central]**, la nueva sublista de búsqueda está a la izquierda o superior:  
**bajo ... alto=central-1**
- Si **clave > a[central]**, la nueva sublista de búsqueda está a la derecha o inferior:  
**bajo = central+1 ... alto**
- El algoritmo termina bien porque se ha encontrado la clave o porque el valor de **bajo** excede al de **alto**, lo que significa que la clave no se encuentra en la lista.

## Búsqueda binaria



## Ejercicio 2

- Lea datos desde un vector y realice la búsqueda binaria.
  - Para datos numéricos
  - Para cadenas



FACULTAD DE  
CIENCIAS

ESCUELA  
PROFESIONAL DE  
CIENCIA DE LA COMPUTACIÓN

FIN

Fundamentos de programación

Clase 04: Ordenación

Profesor: Carlos Diaz

cdiazd@uni.edu.pe

# Contenido

- ¿Qué es un ordenamiento?
- Tipos de ordenación
- Métodos de ordenación directos
  - Ordenación por Selección (Selection Sort)
  - Ordenación por Inserción (Insertion Sort)
  - Ordenación de Burbuja (Bubble Sort)

# ¿Qué es un ordenamiento?

- La ordenación de datos (sort en inglés) es una operación que consiste en reorganizar un conjunto de datos en una secuencia específica respecto a uno o varios los campos.

**Ejemplo:** La guía telefónica tiene los campos **Nombre, Apellidos, Dirección, CP, Ciudad y Teléfono,**

La guía telefónica está organizada en orden alfabético ascendente por **Nombre, Apellidos, etc.**

- **Clave:** Es el campo respecto al cual está ordenado el conjunto de datos.

En el ejemplo el campo **Nombre** es la clave con la que se ordena en una primera instancia.

Páginas-Blancas.org Búsqueda de personas y particulares en Asturias

Nombre	Apellidos	Dirección	CP	Ciudad	Teléfono
Juan	Perez Alvarez	Calle marques de teverga, (Oviedo)	33005	Asturias	985 236 376
Juan	Perez Arango	Avda torcuato fernandez miranda, (Gijon)	33203	Asturias	985 331 183
Juan	Perez Garcia	Barrio villa, (Valdes)	33700	Asturias	985 641 152
Juan	Perez Lomba	Calle horacio fernandez inguanzo, (Gijon)	33209	Asturias	985 155 756
Juan	Perez Morgado	Calle candido fernandez riesgo, (Langreo)	33900	Asturias	985 675 644
Juan	Perez Perez	Mutafalen, (Tineo)	33873	Asturias	985 806 184
Juan	Perez Rios	Calle ruiz, (Gijon)	33213	Asturias	985 325 044
Juan	Perez Rodriguez	Avenida gijon, (Siero)	33420	Asturias	985 267 390
Juan	Perez Villanueva	Ctra general, (Piloña)	33584	Asturias	985 706 353

# Tipos de ordenación

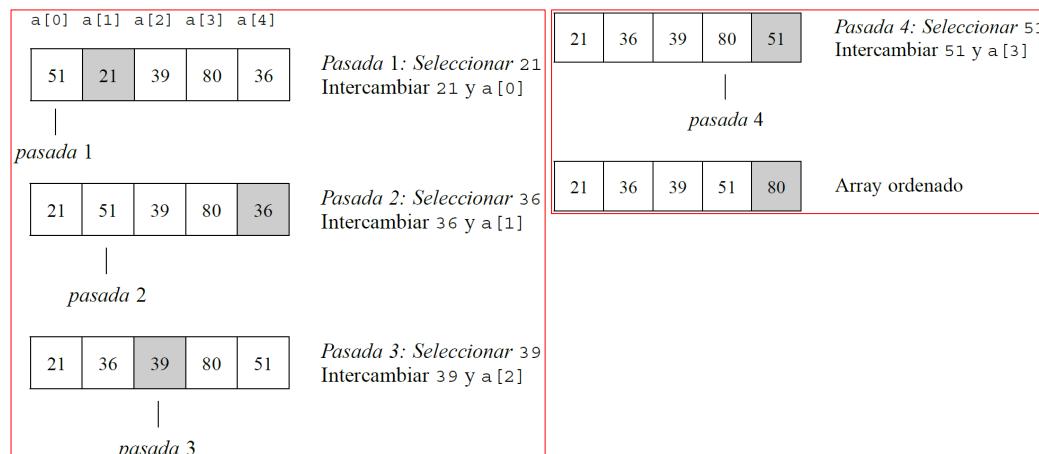
- Se suelen clasificar en tres tipos:
  1. Segundo el orden,
    - **Ascendentes:**  $i < j \Rightarrow K[i] <= K[j]$
    - **Descendentes:**  $i < j \Rightarrow K[i] >= K[j]$
  2. Segundo el lugar donde se almacenen.
    - **Interna:** Cuando los datos se encuentran en la memoria de la computadora.
    - **Externa:** Cuando los datos se encuentran en unidades de almacenamiento externo.
  3. Los métodos de ordenación interna, a su vez, se clasifican según la cantidad de datos.
    - **Directos:** Son eficientes cuando se trata de una pequeña cantidad de datos (Interchange Sort, Selection Sort, Insertion Sort, Bubble Sort, etc.).
    - **Indirectos:** Son eficientes en grandes cantidades de datos (Shell Sort, Quick Sort, Merge Sort, Heap Sort, Radix Sort, etc.).

# Ordenación por Selección

- Su funcionamiento es el siguiente:
  1. Buscar el mínimo elemento de la lista.
  2. Intercambiarlo con el primero.
  3. Buscar el siguiente mínimo en el resto de la lista.
  4. Intercambiarlo con el segundo.
- Y en general:
  1. Buscar el mínimo elemento entre una posición **i** y el final de la lista.
  2. Intercambiar el mínimo con el elemento de la posición **i**.
  3. Si hay **n** elementos, repite **n-1** veces, el último ya no lo compara.

8  
5  
2  
6  
9  
3  
1  
4  
0  
7

# Ordenación por Selección



# Ejercicio 1

- Programe la función `ordSeleccion()` que implemente el algoritmo de Ordenación por Selección.

El tamaño del array y los datos se leen desde teclado.

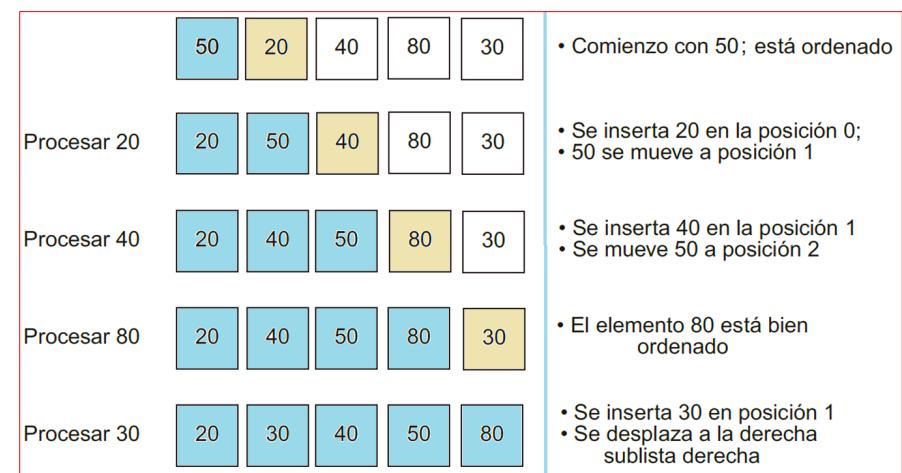
- a) Para datos numéricos
- b) Para cadenas

# Ordenación por Inserción

- Este método de ordenación es similar al proceso típico de ordenar las cartas de una baraja, que consiste en insertar una carta en su posición correcta dentro de una lista que ya está ordenada.
1. El primer elemento a[0] se considera ordenado; es decir, la lista inicial consta de un elemento.
  2. Se inserta a[1] en la posición correcta; delante o detrás de a[0], dependiendo de que sea menor o mayor y así sucesivamente.
  3. Si hay n elementos, repite n-1 veces, ya que el primero se supone que ya está ordenado.

6 5 3 1 8 7 2 4

# Ordenación por Inserción



## Ejercicio 2

- Programe la función `ordInsercion()` que implemente el algoritmo de Ordenación por Inserción.

El tamaño del array y los datos se leen desde teclado.

- a) Para datos numéricos
- b) Para cadenas

## Ordenación por Burbuja

- El método de ordenación por burbuja es el más conocido y popular entre estudiantes y aprendices de programación, por su facilidad de comprender y programar; por el contrario, es el menos eficiente y por ello, normalmente, se aprende su técnica pero no suele utilizarse.
- La técnica utilizada se denomina ordenación por burbuja debido a que los valores más grandes (**pequeños**) “**burbujean**” suben gradualmente hacia la parte superior del array de modo similar a como suben las burbujas en el agua, mientras que los valores menores (**mayores**) se hunden en la parte inferior del array.

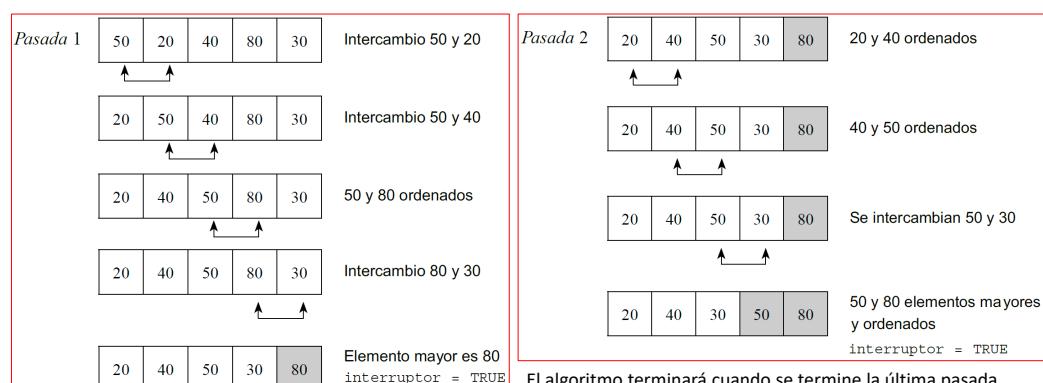
## Ordenación por Burbuja

6 5 3 1 8 7 2 4

## Ordenación por Burbuja

- Las etapas del algoritmo son:
- En la pasada 1 se comparan elementos adyacentes.  
 $(a[0],a[1]),(a[1],a[2]),(a[2],a[3]),\dots(a[n-2],a[n-1])$   
Se realizan  $n - 1$  comparaciones, por cada pareja  $(a[i],a[i+1])$   
Se intercambian los valores si  $a[i+1] < a[i]$ .  
Al final de la pasada, el elemento mayor de la lista está situado en  $a[n-1]$ .
- En la pasada 2 se realizan las mismas comparaciones e intercambios, terminando con el elemento segundo mayor valor en  $a[n-2]$  .
- El proceso termina con la pasada  $n - 1$ , en la que el elemento más pequeño se almacena en  $a[0]$ .
- El algoritmo tiene una mejora inmediata, el proceso de ordenación puede terminar en la pasada  $n - 1$ , o bien antes, si en un una pasada no se produce intercambio alguno entre elementos del array es porque ya está ordenado, entonces no es necesario más pasadas.

## Ordenación por Burbuja



El algoritmo terminará cuando se termine la última pasada ( $n - 1$ ), o bien cuando el valor del interruptor sea **falso**, es decir no se haya hecho ningún intercambio.

## Ejercicio 3

- Programe la función `ordBurbuja()` mejorado que implemente el algoritmo de Ordenación por Burbuja.  
El tamaño del array y los datos se leen desde teclado.
  - Para datos numéricos
  - Para cadenas

## Contenido

- Mezcla de listas ordenadas
- Métodos de ordenación indirectos
  - Ordenación por mezcla (Merge Sort)
  - Ordenación rápida (Quick Sort)

FIN

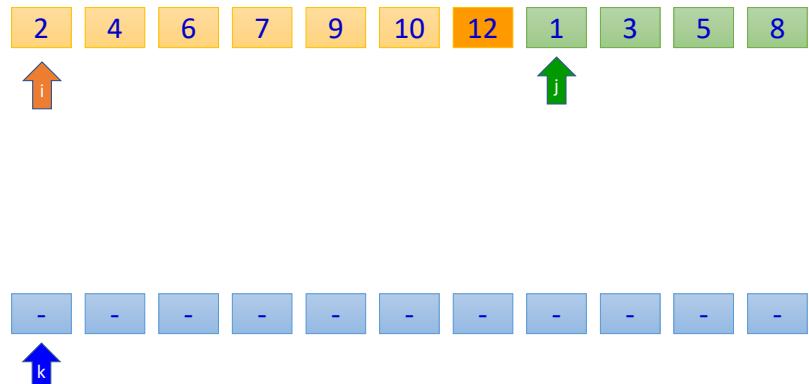
# Mezcla de listas ordenadas

- El algoritmo de mezcla utiliza un array auxiliar, `temporal[]` para realizar la fusión entre dos sublistas ordenadas, que se encuentran en el mismo vector `a[]`, delimitadas por los índices `Izq`, `Centro` y `Der`.



- A partir de estos índices se pueden recorrer las sublistas `a[Izq...Centro]` y `a[Centro+1...Der]`
- En cada pasada del algoritmo de mezcla se compara `a[i]` con `a[j]`, el menor o igual se copia en el vector auxiliar, `temporal[]`, y avanzan el subíndice del que se copio y también el subíndice del vector auxiliar.
- Los elementos que sobran se copian al final del vector `temporal[]`.
- Y finalmente el vector `temporal[]` se copia al vector `a[]`.

## Simulación Mezcla de listas ordenadas



## Ejercicio 1

- Programe la función `mezclaLista()` que implemente el algoritmo de mezcla de listas ordenadas.

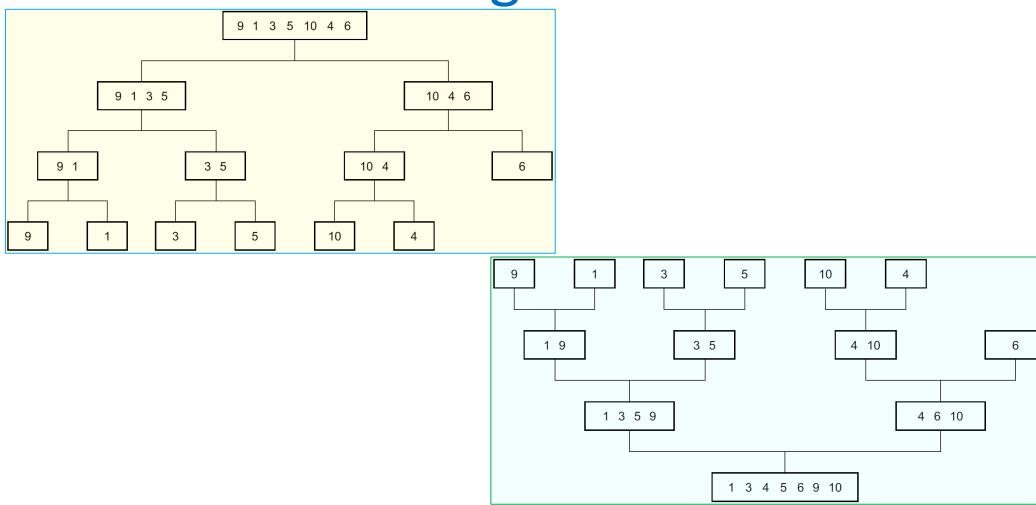
El tamaño del array y los datos se leen desde teclado.

## Merge Sort

- Este método de ordenación divide la lista por la posición central, vuelve a dividir cada una de estas partes en dos mitades y así sucesivamente hasta tener a un solo elemento, la que en si misma esta ordenada.
- Al regresar por la función recursiva utiliza un [algoritmo de mezcla de listas ordenadas](#) y fusiona cada par de sublistas, obteniendo un sublista ordenada.
- Repite el proceso hasta tener la lista ordenada.
- **Ejemplo:** Ordenar la siguiente lista.  $\text{Centro} = (\text{Izq} + \text{Der}) / 2 = (0 + 6) / 2 = 3$ , es decir el **centro** es  $\text{lista}[3] = 5$ .



# Merge Sort



# Ejercicio 2

- Programe la función `ordMerge()` que implemente el algoritmo de ordenación Merge.

El tamaño del array y los datos se leen desde teclado.

# Ordenación rápida (QuickSort)

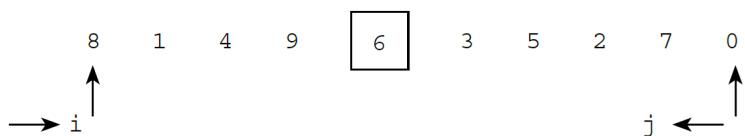
- Fue creado por Charles Antony Richard Hoare en 1960.
- El algoritmo divide los  $n$  elementos de la lista a ordenar en dos partes o particiones separadas por un elemento: una **partición izquierda**, un **elemento central** denominado **pivote**, y una **partición derecha**.
- La partición se hace de tal forma que todos los elementos de la primera sublistas (partición izquierda) son menores que todos los elementos de la segunda sublistas (partición derecha).
- Las dos sublistas se ordenan entonces independientemente.
- Para dividir la lista en particiones (sublistas) se elige uno de los elementos de la lista como pivote o elemento de partición.
- Si los elementos de la lista están en orden aleatorio, se puede elegir cualquier elemento como pivote, por ejemplo, el elemento central de la lista.

# Ordenación rápida (QuickSort)

- Si la lista tiene algún orden parcial conocido, se puede tomar otra decisión para el pivote.
- Idealmente, el pivote se debe elegir de modo que se divida la lista por la mitad.
- Una vez que el pivote ha sido elegido, se utiliza para ordenar el resto de la lista en dos sublistas: una tiene todos los números menores que el pivote y la otra todos los números mayores o iguales que el pivote.
- Estas dos listas parciales se ordenan recursivamente utilizando el mismo algoritmo; es decir, se llama sucesivamente al propio algoritmo quicksort.
- La lista final ordenada se consigue concatenando la primera sublistas, el pivote y la segunda sublistas, en ese orden, en una única lista.
- La primera etapa de quicksort es la división o “particionado” recursivo de la lista hasta que todas las sublistas consten de sólo un elemento.

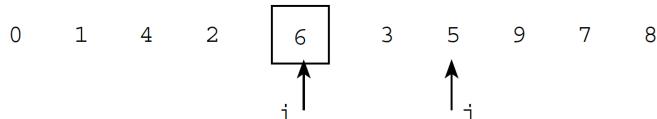
# Algoritmo ordenación QuickSort

- **Lista original:** 8 1 4 9 6 3 5 2 7 0
- **Pivote (elemento central)** 6 = vector[primero+ultimo]/2 donde ultimo=n-1, n= elementos del vector
- Una vez elegido el pivote, la segunda etapa requiere mover todos los elementos menores al pivote a la parte izquierda del array y los elementos mayores o iguales a la parte derecha. Para ello se recorre la lista de izquierda a derecha utilizando un índice i, que se inicializa a la posición más baja (inferior), buscando un elemento mayor al pivote. También se recorre el array de derecha a izquierda buscando un elemento menor. Para esto se utilizará el índice j, inicializado a la posición más alta (superior).
- El índice i se detiene en el elemento 8 (mayor que el pivote) y el índice j se detiene en el elemento 0 (menor que el pivote).

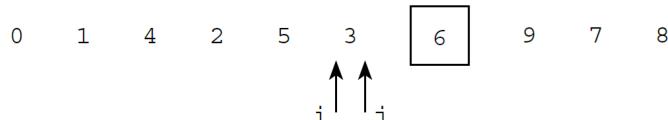


# Algoritmo ordenación QuickSort

- Se intercambian los elementos mientras que i y j no se crucen. En caso contrario se detiene este bucle. En el caso anterior se intercambian 9 y 2.
- Continúa la exploración y ahora el contador i se detiene en el elemento 6 (que es el pivote) y el índice j se detiene en el elemento menor 5.

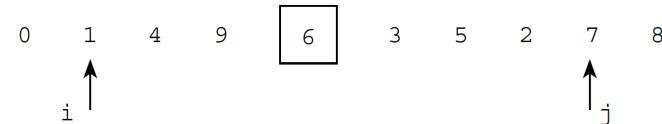


- Se intercambian. Los índices se mueven y tienen actualmente los valores i = 5, j = 5. Continúa la exploración mientras  $i \leq j$ .

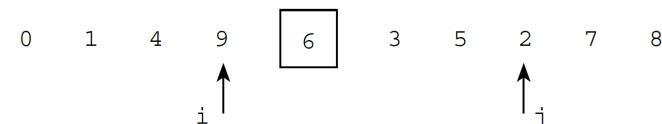


# Algoritmo ordenación QuickSort

- Ahora, se intercambian a[i] y a[j] para que estos dos elementos se sitúen correctamente en cada sublista; y se incrementa el índice i, y se decrementa j para seguir los intercambios.

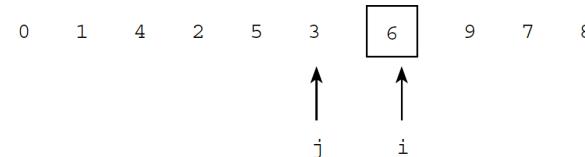


- A medida que el algoritmo continúa, i se detiene en el elemento mayor, 9, y j se detiene en el elemento menor, 2.



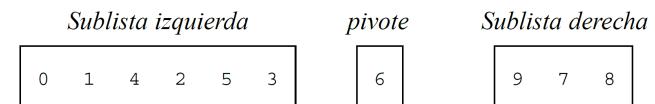
# Algoritmo ordenación QuickSort

- Continúa la exploración y ahora el contador i se detiene en el elemento 6 (que es el pivote) y el índice j se detiene en el elemento menor 5. Acaba con i = 6, j = 5, pues  $i > j$ .



- En esta posición los índices i y j han cruzado posiciones en el array, se detiene la búsqueda y no se realiza ningún intercambio ya que el elemento al que accede j está ya correctamente situado.

- Las dos sublistas ya han sido creadas, la lista original se ha dividido en dos particiones:



## Ejemplo QuickSort

0	1	2	3	4	5	6	7	8	9	10	11
79	21	15	99	88	65	75	85	76	46	84	24

FIN

## Ejercicio 3

- Programe la función `ordQuickSort()` que implemente el algoritmo de ordenación QuickSort.

El tamaño del array y los datos se leen desde teclado.



# FUNDAMENTOS DE PROGRAMACIÓN

Punteros I

Profesor: Carlos Diaz

cdiazd@uni.edu.pe

# Punteros I

- Concepto de puntero
- El operador de dirección & y el operador de desreferenciación \*
- Punteros en operaciones de asignación
- El operador new (variables dinámicas)
- El operador delete
- Arrays dinámicos unidimensionales
- Arrays dinámicos multidimensionales

## Concepto de puntero

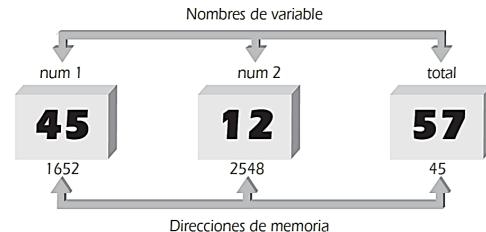
- Un **puntero** es la dirección en memoria de una variable. Por ejemplo:

num1=45;

num2=12;

total=num1+num2;

El puntero de la  
variable num1 es 1652



- Un puntero se puede guardar en una variable, llamada **variable puntero**.
- Sin embargo, aunque un puntero es una dirección de memoria y una dirección de memoria es un número, no podemos guardar un puntero en una variable de tipo *int* o *double*.
- La variable que va a contener un puntero se debe declarar como de tipo puntero. Por ejemplo:

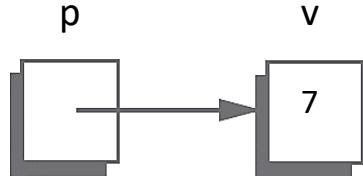
double \*p;

- En este caso, se dice que **p** es la dirección de una variable que contiene un *double*, es decir "la variable puntero **p** apunta a una variable tipo *double*".

## El operador & y \*

- El operador de dirección & se utiliza para obtener la dirección de una variable.
- El operador de desreferenciación \* reproduce la variable a la que apunta la variable puntero.

```
#include<iostream>
using namespace std;
int main(){
    int *p, v;
    v=7;
    p=&v; // p es la dirección de la variable v
    *p=*p+v+6; // v y *p se refieren a la misma variable
    cout<<"Direccion de v: "<<p<<endl;
    cout<<"Direccion de v: "<<&v<<endl;
    cout<<"Valor de v: "<<*p<<endl;
    cout<<"Valor de v: "<<v<<endl;
    cout<<"Direccion de p: "<<&p<<endl;
    return 0;
}
```



## Punteros en operaciones de asignación

- Podemos asignar el valor de una variable de puntero a otra variable de puntero. Esto copia una dirección de una variable de puntero a la otra. Sean los punteros p1 y p2:

```
#include<iostream>
using namespace std;
```

```
int main() {
```

```
    int *p1, *p2, v1, v2;
```

```
    v1 = 84;
```

```
    v2 = 99;
```

```
    p1 = &v1;
```

```
    p2 = &v2;
```

```
    cout << "Contenido de p1 (direccion de v1) " << p1 << endl;
```

```
    cout << "Contenido de p2 (direccion de v2): " << p2 << endl;
```

```
    p1=p2;
```

```
    cout << "Direccion de v1: " << p1 << endl;
```

```
    cout << "Direccion de v2: " << p2 << endl;
```

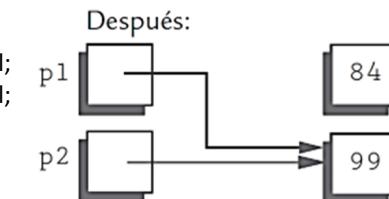
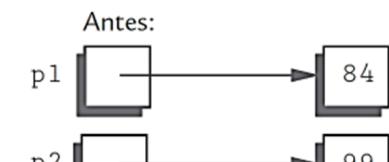
```
    cout << "Valor a donde apunta p1: " << *p1 << endl;
```

```
    cout << "Valor a donde apunta p2: " << *p2 << endl;
```

```
    cout << "v1: "<<v1<<endl;
```

```
    cout << "v2: "<<v2<<endl;
```

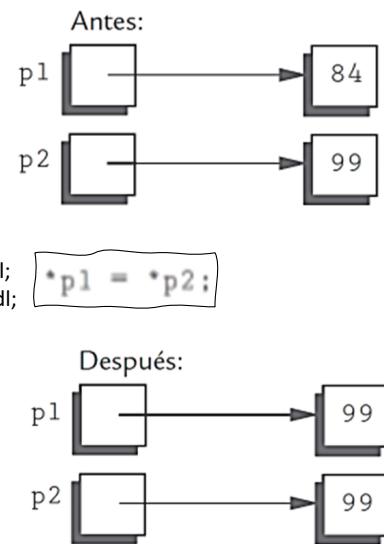
```
    return 0;
}
```



# Punteros en operaciones de asignación

- Podemos asignar el valor del contenido de lo que apunta una variable de puntero al contenido otra variable de puntero. Esto copia el contenido de lo que apunta una variable de puntero a la otra. Sean los punteros p1 y p2:

```
#include<iostream>
using namespace std;
int main() {
    int *p1, *p2, v1, v2;
    v1 = 84;
    v2 = 99;
    p1 = &v1;
    p2 = &v2;
    cout << "Contenido de p1 (direccion de v1) " << p1 << endl;
    cout << "Contenido de p2 (direccion de v2): " << p2 << endl;
    *p1=*p2;
    cout << "Direccion de v1: " << p1 << endl;
    cout << "Direccion de v2: " << p2 << endl;
    cout << "Valor a donde apunta p1: " << *p1 << endl;
    cout << "Valor a donde apunta p2: " << *p2 << endl;
    cout << "v1: "<<v1<<endl;
    cout << "v2: "<<v2<<endl;
    return 0;
}
```



# El operador new (variables dinámicas)

- Hemos visto que podemos usar un de puntero `p` para referirnos a una variable con identificador (con nombre, por ejemplo `v`).
- También podemos manipular variables aunque éstas carezcan de identificadores.
- Usando el operador `new` se puede crear variables sin identificadores, haciendo referencia a estas variables sin nombre mediante punteros.
- Por ejemplo, lo siguiente crea una nueva variable de tipo `int` y asigna a la variable puntero `p` la dirección de esta nueva variable (es decir, `p` apunta a esta nueva variable sin nombre):

```
int *p;  
p = new int;
```

- Podemos usar `*p` para referirnos a esta nueva variable. Podemos hacer con esta variable sin nombre lo mismo que con cualquier otra variable de tipo `int`. Por ejemplo:

```
cin >> *p;  
*p = *p + 7;  
cout << *p;
```

Las variables que se crean con el operador `new` se llaman **variables dinámicas** porque se crean y se destruyen mientras el programa se está ejecutando.

# El operador delete

- Toda variable dinámica nueva creada por un programa consume parte de la memoria de la computadora.
- El sistema operativo reserva un área especial de la memoria, llamada **montículo (heap)** para las variables dinámicas.
- Si nuestro programa crea demasiadas variables dinámicas, consumirá toda la memoria del montículo. Si esto sucede, todas las llamadas subsecuentes a `new` fracasarán.
- Si nuestro programa ya no necesita una variable dinámica dada, la memoria ocupada por esa variable puede ser liberada.
- El operador `delete` elimina una variable dinámica a la que apunta la variable puntero y devuelve al montículo la memoria que ocupaba, para poder reutilizarla.
- Supongamos que `p` es una variable de puntero que está apuntando a una variable dinámica. Lo que sigue destruye la variable dinámica a la que `p` apunta y devuelve al almacén libre la memoria ocupada por esa variable dinámica:

```
delete p;
```

- Después de la llamada a `delete`, el valor de `p` es indefinido o sea no sabemos a donde esta apuntando ni que valor hay en lugar que esta apuntando.

# Ejemplo

```
#include<iostream>
using namespace std;
int main(){
    cout<<"Asigna memoria"<<endl;
    int *p;
    p=new int(7); //p1 apunta a una nueva variable dinamica
    cout << p<<endl; //Dirección de la variable dinámica, la que tiene al 7
    cout << *p<<endl; //El valor de la variable dinámica
    cout << &p<<endl; //Dirección de p
    cout<<"Libera memoria"<<endl;
    delete p; //Destruye la variable dinamica, liberando memoria
    cout << p<<endl; //Dirección de la variable dinamica
    cout << *p<<endl; //El valor de la variable dinámica
    cout << &p<<endl; //Dirección de p
    return 0;
}
```

# Arrays dinámicos unidimensionales

- Un problema que presentan los arreglos que hemos usado hasta ahora es que es preciso especificar el tamaño del arreglo en el momento de escribir el programa, y podría ser imposible saber de qué tamaño necesitamos un arreglo antes de ejecutar el programa.
- Los arreglos dinámicos evitan estos problemas.
- Los arreglos dinámicos se crean con el operador `new`.
- Por ejemplo, lo siguiente crea una variable arreglo dinámica con `n` elementos de tipo double.

```
double *p;  
cin>>n;  
p = new double[n];
```

- La instrucción `delete` para un arreglo dinámico es similar a la que vimos antes, excepto que en este caso es preciso incluir un par de corchetes, así:

```
delete [] p;
```

# Arrays dinámicos multidimensionales

- Podemos tener arreglos dinámicos multidimensionales. Sólo recuerde que los arreglos multidimensionales son **arreglos de arreglos**, o **arreglos de arreglos de arreglos**, etcétera.
- Por ejemplo, para crear un arreglo dinámico bidimensional, debemos recordar que éste es un arreglo de arreglos unidimensionales.
- Para obtener un arreglo de enteros de  $3 \times 4$ :

```
int **m ;  
m= new int*[3];
```

- Éste es un arreglo de tres punteros, cada uno de los cuales puede apuntar a un arreglo dinámico de enteros, como sigue:

```
for (int i = 0; i < 3; i++)  
    m[i] = new int[4];
```

- El arreglo resultante `m` es un arreglo dinámico de 3 por 4.
- Para liberar la memoria primero liberamos las filas y luego la matriz.

## Ejemplo

```
#include <iostream>  
using namespace std;  
int main(){  
    int d1, d2,i,j;  
    cout << "Escriba las dimensiones de las filas y de las columnas del arreglo:\n";  
    cin >> d1 >> d2;  
    int **m = new int*[d1];  
    for (i = 0; i < d1; i++)  
        m[i] = new int[d2];  
    //m es ahora arreglos a d1 de d2.  
    cout << " Escriba " << d1 << " filas de " << d2 << " enteros cada una:\n";  
    for (i = 0; i < d1; i++)  
        for (j = 0; j < d2; j++)  
            cin >> m[i][j];  
    cout << "Valores del arreglo bidimensional:\n";  
    for (i = 0; i < d1; i++)  
    {  
        for (j = 0; j < d2; j++)  
            cout << m[i][j] << " ";  
        cout << endl;  
    }  
    //Liberando la memoria  
    for (i = 0; i < d1; i++)  
        delete[] m [i];  
    delete[] m;  
    return 0;  
}
```

FIN



# FUNDAMENTOS DE PROGRAMACIÓN

## Punteros II

Profesor: Carlos Diaz

cdiazd@uni.edu.pe

## Punteros II

- Concepto de referencia
- Diferentes usos de & en C++
- Parámetros por valor y por referencia
- Punteros NULL y void
- Punteros y arrays
- Puntero a puntero (puntero doble)
- Aritmética de punteros
- Puntero en los arrays de dos dimensiones
- Paso de vectores y matrices a funciones

## Concepto de referencia

- Cuando una variable se declara, se asocian tres atributos a la misma: su **nombre (identificador)**, su **tipo** y su **dirección en memoria**.
- Una **referencia** es un alias de otra variable. Se declara utilizando el operador de referencia (&).
- El identificador de la variable y referencia son nombres diferentes para la misma variable.

## Ejemplo 1

```
#include <iostream>
using namespace std;
int main() {
    int v1 = 75, v2 = 15; // Declaración de variable e inicialización
    int& ref1 = v1, & ref2 = v2; //Referencia e inicialización
    cout << " Contenido de v1 = " << v1 << endl;
    cout << " Contenido de v2 = " << v2 << endl;
    cout << " Dirección de v1 = " << &v1 << endl;
    cout << " Dirección de v2 = " << &v2 << endl;
    cout << " Contenido de ref1 = " << ref1 << endl;
    cout << " Contenido de ref2 = " << ref2 << endl;
    cout << " Dirección de ref1 = " << &ref1 << endl;
    cout << " Dirección de ref2 = " << &ref2 << endl;
    return 0;
}
```

The terminal window displays the following output:

```
Contenido de v1 = 75
Contenido de v2 = 15
Dirección de v1 = 0x6ffd9c
Dirección de v2 = 0x6ffd98
Contenido de ref1 = 75
Contenido de ref2 = 15
Dirección de ref1 = 0x6ffd9c
Dirección de ref2 = 0x6ffd98
```

## Diferentes usos de & en C++

- El carácter & tiene diferentes usos en C++:
- Cuando se utiliza como prefijo de un nombre de una variable, devuelve la dirección de esa variable.

```
cout << "Direccion de la variable= " << &var << endl;
```
- Cuando se utiliza como un sufijo en una declaración de una variable, declara la variable como sinónimo de la variable a la que se ha inicializado.

```
int& ref = var; //Referencia e inicialización
```
- Cuando se utiliza como sufijo en una declaración de parámetros de una función, declara el parámetro como referencia a la variable que se pasa a la función.

```
int funcion(int &n, int &m); // n y m pasan como referencia
```

## Parámetros por valor y por referencia

- Hasta ahora, la forma en que hemos declarado y pasado los parámetros de las funciones se conoce como "**por valor**". Esto quiere decir que cuando el control pasa a la función, los valores de los parámetros en la llamada se copian a las variables locales de la función.

```
#include<iostream>
using namespace std;
int funcion(int n, int m);
int main() {
    int a, b;
    a = 10;
    b = 20;
    cout << "a,b = " << a << ", " << b << endl;
    cout << "funcion(a,b) =" << funcion(a, b) << endl;
    cout << "a,b = " << a << ", " << b << endl;
    return 0;
}
int funcion(int n, int m) {
    n = n + 1;
    m = m + 2;
    return n + m;
}
```

## Parámetros por valor y por referencia

- Si queremos que los cambios realizados en los parámetros dentro de la función se conserven al retornar de la llamada, deberemos pasarlos "**por referencia**". Esto se hace declarando los parámetros de la función como referencias a las variables.

```
#include<iostream>
using namespace std;
int funcion(int &n, int &m);
int main() {
    int a, b;
    a = 10;
    b = 20;
    cout << "a,b = " << a << ", " << b << endl;
    cout << "funcion(a,b) =" << funcion(a, b) << endl;
    cout << "a,b = " << a << ", " << b << endl;
    return 0;
}
int funcion(int &n, int &m) {
    n = n + 1;
    m = m + 2;
    return n + m;
}
```

## Parámetros por valor y por referencia

- También podemos pasar directamente las direcciones de la variable a la función.

```
#include<iostream>
using namespace std;
int funcion(int* n, int* m);
int main() {
    int a, b;
    a = 10;
    b = 20;
    cout << "a,b = " << a << ", " << b << endl;
    cout << "funcion(a,b) =" << funcion(&a, &b) << endl;
    cout << "a,b = " << a << ", " << b << endl;
    return 0;
}
int funcion(int* n, int* m) {
    *n = *n + 1;
    *m = *m + 2;
    return *n + *m;
}
```

## Punteros NULL y void

- Un puntero nulo no apunta a ningún dato válido, se utiliza para proporcionar a un programa un medio de conocer cuando una variable puntero no apunta a un dato válido.
- Los punteros void pueden apuntar a cualquier tipo de dato.

Ejemplo:

```
int x, *px = &x, &rx = x;
float *z = NULL;
void *r = &x, *t = z;
```

- **x** es una variable entera
- **px** es un puntero a una variable entera inicializado con la dirección de **x**
- **rx** es una referencia a un entero inicializada con el valor de **x**.
- **z** es un puntero a un real inicializado a **NULL**.
- **r** es un puntero void inicializado a un puntero a entero,
- **t** es un puntero void inicializado a un puntero a float.

## Aritmética de punteros

- A un puntero se le puede sumar o restar un entero n; esto hace que apunte n posiciones adelante, o atrás de la actual.
- A una variable puntero se le puede aplicar el operador ++, o el operador --. Esta operación hace que el operador contenga la dirección del siguiente, o anterior elemento.
- No tiene sentido sumar o restar una constante de coma flotante a un puntero.
- **Operaciones no válidas con punteros:** no se puede sumar dos punteros; no se puede multiplicar dos punteros; no se puede dividir dos punteros.

## Punteros y arrays

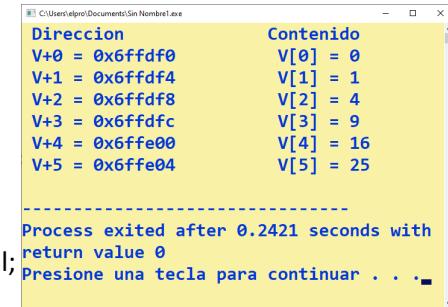
- Los arrays y los punteros están fuertemente relacionados en el lenguaje C++.
- El nombre de un array es un puntero que contiene la dirección en memoria del primer elemento de la secuencia de elementos que forman el array.
- Este nombre del array es un puntero constante ya que no se puede modificar, sólo se puede acceder para indexar a los elementos del array.
- Para visualizar, almacenar o calcular un elemento de un array, se puede utilizar notación de subíndices o notación de punteros, a un puntero p se le puede sumar un entero n, desplazándose el puntero tantos bytes como ocupe el tipo de dato.
- Si se tiene la siguiente declaración de array int V[6] = {1, 11, 21, 31, 41, 51};, su almacenamiento en memoria será el siguiente:

	V[0]	V[1]	V[2]	V[3]	V[4]	V[5]
memoria	1	11	21	31	41	51
	*V	*(V + 1)	*(V + 2)	*(V + 3)	*(V + 4)	*(V + 5)

## Ejemplo 2

- Inicialización y visualización de un array con punteros.
- El programa inicializa un array de reales y visualiza las direcciones de cada una de las posiciones así como sus contenidos.

```
#include <iostream>
using namespace std;
int main() {
    float V[6];
    for (int i = 0; i < 6; i++)
        *(V + i) = i * i; //cuadrado de i
    cout << " Direccion\t\tContenido"
        << endl;
    for (int i = 0; i < 6; i++)
    {
        cout << " V+" << i << " = " << V + i;
        cout << "\t\t V[" << i << "] = " << *(V + i) << "\n";
    }
}
```



Direccion	Contenido
V+0 = 0x6ffffd0	V[0] = 0
V+1 = 0x6ffffd4	V[1] = 1
V+2 = 0x6ffffd8	V[2] = 4
V+3 = 0x6ffffdc	V[3] = 9
V+4 = 0x6ffe000	V[4] = 16
V+5 = 0x6ffe004	V[5] = 25

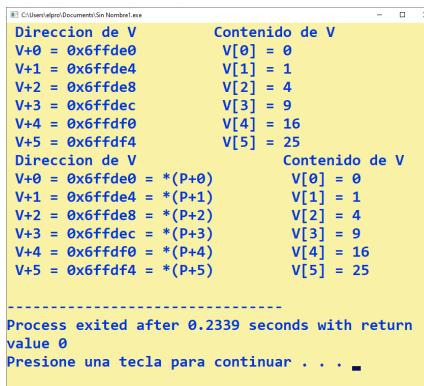
-----

Process exited after 0.2421 seconds with  
return value 0  
Presione una tecla para continuar . . .

## Ejemplo 3

- Se puede declarar un array de punteros, como un array que contiene punteros como elementos, cada uno de los cuales apuntará a otro dato específico.
- El siguiente programa inicializa el array de reales V, así como el array de punteros a reales P, con las direcciones de las sucesivas posiciones del array V. Luego, visualiza las direcciones y los contenidos de V usando el array de punteros P.

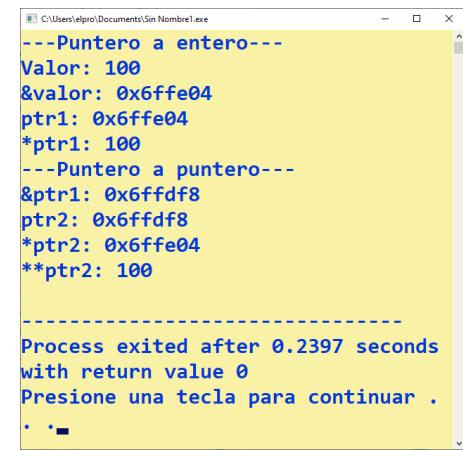
```
#include <iostream>
using namespace std;
int main()
{
    float V[6], * P[6];
    for (int i = 0; i < 6; i++)
    {
        *(V + i) = i * i;
        *(P + i) = V + i; // inicialización de array de punteros
    }
    cout << " Dirección de V\t\tContenido de V" << endl;
    for (int i = 0; i < 6; i++)
    {
        cout << " V+" << i << "=" << (V + i);
        cout << "\t\tV[" << i << "] = " << *(V + i) << "\n";
    }
    cout << " Dirección de V\t\tContenido de V" << endl;
    for (int i = 0; i < 6; i++)
    {
        cout << " V+" << i << "=" << *(P + i) << "= *(P+" << i << ")";
        cout << "\t\tV[" << i << "] = " << **(P + i) << "\n";
    }
}
```



## Puntero a puntero

- Un puntero puede apuntar a otra variable puntero.
- Para declarar un puntero a un puntero se hace preceder a la variable con dos asteriscos (\*\*).
- En el siguiente código, ptr2 es un puntero a un puntero.

```
#include <iostream>
using namespace std;
int main()
{
    int valor = 100;
    int* ptr1 = &valor; //Puntero a un entero
    int** ptr2 = &ptr1; //Puntero a un puntero
    cout << "---Puntero a entero---" << endl;
    cout << "Valor: " << valor << endl;
    cout << "&valor: " << &valor << endl;
    cout << "ptr1: " << ptr1 << endl;
    cout << "*ptr1: " << *ptr1 << endl;
    cout << "---Puntero a puntero---" << endl;
    cout << "&ptr1: " << &ptr1 << endl;
    cout << "ptr2: " << ptr2 << endl;
    cout << "*ptr2: " << *ptr2 << endl;
    cout << "***ptr2: " << **ptr2 << endl;
}
```



## Puntero en los arrays de dos dimensiones

- Para apuntar a un array bidimensional como tal, o lo que es lo mismo, para apuntar a su inicio, el compilador de C++ considera que un array bidimensional es en realidad un array de punteros a los arrays que forman sus filas.
- Por tanto, será necesario un puntero doble o puntero a puntero, que contendrá la dirección del primer puntero del array de punteros a cada una de las filas del array bidimensional o matriz.
- Si a se ha definido como un array bidimensional, el nombre del array a es un puntero constante que apunta a la primera fila a[0]. El puntero a+1 apunta a la segunda fila a[1], etc. A su vez a[0] es un puntero que apunta al primer elemento de la fila 0 que es a[0][0]. El puntero a[1] es un puntero que apunta al primer elemento de la fila 1 que es a[1][0], etc.

## Ejemplo 4

- Dada la declaración `float A[5][3]` que define un array bidimensional de cinco filas y tres columnas, se tiene la siguiente estructura:

Puntero a puntero fila	Puntero a fila	ARRAY BIDIMENSIONAL float A[4][3]
A	→ A[0]	A[0][0] A[0][1] A[0][2]
A+1	→ A[1]	A[1][0] A[1][1] A[1][2]
A+2	→ A[2]	A[2][0] A[2][1] A[2][2]
A+3	→ A[3]	A[3][0] A[3][1] A[3][2]
A+4	→ A[5]	A[4][0] A[4][1] A[4][2]

- A es un puntero que apunta a un array de 5 punteros A[0], A[1], A[2], A[3], A[4].
- A[0] es un puntero que apunta a un array de 3 elementos A[0][0], A[0][1], A[0][2].
- A[1] es un puntero que apunta a un array de 3 elementos A[1][0], A[1][1], A[1][2].
- A[2] es un puntero que apunta a un array de 3 elementos A[2][0], A[2][1], A[2][2].
- A[3] es un puntero que apunta a un array de 3 elementos A[3][0], A[3][1], A[3][2].
- A[4] es un puntero que apunta a un array de 3 elementos A[4][0], A[4][1], A[4][2].

## Ejemplo 4 (continuación)

- $A[i][j]$  es equivalente a las siguientes expresiones:

$*(A[i]+j)$  :  $A[i]$  es el nombre del vector.

$*(*A+i)+j$  : Donde cambiamos  $A[i]$  por  $*(A+i)$ .

$*(&A[0][0]+3*i+j)$  : Donde se hizo el siguiente cálculo:

- $A$  es un puntero que apunta a  $A[0]$ .

- $A[0]$  es un puntero que apunta a  $A[0][0]$ .

- Si  $A[0][0]$  se encuentra en la dirección de memoria 100 y teniendo en cuenta que un float ocupa 4 bytes, la siguiente tabla muestra un esquema de la memoria:

Contenido de puntero a puntero fila	Contenido de puntero a fila	Direcciones del array bidimensional float A[4][3]		
$*A = A[0]$	$A[0] = 100$	$\&A[0][0] = 100$	$\&A[0][1] = 104$	$\&A[0][2] = 108$
$*(A+1) = A[1]$	$A[1] = 112$	$\&A[1][0] = 112$	$\&A[1][1] = 116$	$\&A[1][2] = 120$
$*(A+2) = A[2]$	$A[2] = 124$	$\&A[2][0] = 124$	$\&A[2][1] = 128$	$\&A[2][2] = 132$
$*(A+3) = A[3]$	$A[3] = 136$	$\&A[3][0] = 136$	$\&A[3][1] = 140$	$\&A[3][2] = 144$
$*(A+4) = A[4]$	$A[5] = 148$	$\&A[4][0] = 148$	$\&A[4][1] = 152$	$\&A[4][2] = 156$

- $*(*A+3*i+j)$  : Donde  $*A$  contiene la dirección de  $A[0][0]$

## Ejemplo 4 Código

```
#include <iostream>
using namespace std;
int main() {
    int A[5][3] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14 };
    cout << "Impresion con indices A[i][j]" << endl;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << A[i][j] << "\t";
        cout << "\n";
    }
    cout << "Impresion con *(A[i]+j)" << endl;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << *(A[i] + j) << "\t";
        cout << "\n";
    }
    cout << "Impresion con *(&A[0][0]+3*i+j)" << endl;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << *(&A[0][0] + 3 * i + j) << "\t";
        cout << "\n";
    }
    cout << "Impresion con *(*A + 3*i + j)" << endl;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << *(*A + 3 * i + j) << "\t";
        cout << "\n";
    }
    return 0;
}
```

## Ejemplo 4 Resultado

## Ejemplo 5

- Direcciones ocupadas por punteros asociados a una matriz.
- El programa muestra las direcciones ocupadas por todos los elementos de una matriz de números reales de 5 filas y 3 columnas, así como las direcciones de los primeros elementos de cada una de las filas, accedidos por un puntero a fila.

```
#include <iostream>
using namespace std;
int main() {
    int A[5][3] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14 };
    cout << " Direcciones de todos los elementos de la matriz\n";
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 3; j++)
            cout << " &A[" << i << "[" << j << "]=" << &A[i][j] << "\t";
    cout << "\n";
    cout << "\n Direcciones de comienzo de las filas de la matriz\n";
    for (int i = 0; i < 5; i++)
        cout << " A[" << i << "] = " << A[i]
            << " contiene la dirección de &A[" << i << "[" << 0 << "]"
            << endl;
    return 0;
}
```

## Ejemplo 5 Resultado

```
C:\Users\elpro\Documents\Sin Nombre1.exe

Direcciones de todos los elementos de la matriz
&A[0][0]=0x6ffdc0    &A[0][1]=0x6ffdc4    &A[0][2]=0x6ffdc8
&A[1][0]=0x6ffdcc   &A[1][1]=0x6ffdd0    &A[1][2]=0x6ffdd4
&A[2][0]=0x6ffdd8   &A[2][1]=0x6ffdc    &A[2][2]=0x6ffde0
&A[3][0]=0x6ffde4   &A[3][1]=0x6ffde8    &A[3][2]=0x6ffdec
&A[4][0]=0x6ffdf0   &A[4][1]=0x6ffdf4    &A[4][2]=0x6ffdf8

Direcciones de comienzo de las filas de la matriz
A[0] = 0x6ffdc0 contiene la dirección de &A[0][0]
A[1] = 0x6ffdcc contiene la dirección de &A[1][0]
A[2] = 0x6ffdd8 contiene la dirección de &A[2][0]
A[3] = 0x6ffde4 contiene la dirección de &A[3][0]
A[4] = 0x6ffdf0 contiene la dirección de &A[4][0]

-----
Process exited after 0.3981 seconds with return value 0
Presione una tecla para continuar . . .
```

## Vector tamaño constante

```
#include <iostream>
using namespace std;
void escribirVector(int A[], int f) {
    int i, j;
    cout << "Vector ingresado\n";
    for (i = 0; i < f; i++)
        cout << A[i] << endl;
}
void leerVector(int A[], int f) {
    int i, j;
    cout << "Ingrese un vector de 3 elementos\n";
    for (i = 0; i < f; i++)
        cin >> A[i];
}
int main() {
    const int F = 3;
    int A[F];
    leerVector(A, F);
    escribirVector(A, F);
}
```

## Vector tamaño variable

```
#include <iostream>
using namespace std;
void escribirVector(int A[], int f) {
    int i, j;
    cout << "Vector ingresado\n";
    for (i = 0; i < f; i++)
        cout << A[i] << endl;
}
void leerVector(int A[], int f) {
    int i, j;
    cout << "Ingrese un vector de n elementos\n";
    for (i = 0; i < f; i++)
        cin >> A[i];
}
int main() {
    int f;
    cout << "Ingrese el tamaño del vector: ";
    cin >> f;
    int *A = new int[f];
    leerVector(A, f);
    escribirVector(A, f);
}
```

## Vector tamaño variable

```
#include <iostream>
using namespace std;
void escribirVector(int *A, int f) {
    int i, j;
    cout << "Vector ingresado\n";
    for (i = 0; i < f; i++)
        cout << A[i] << endl;
}
void leerVector(int *A, int f) {
    int i, j;
    cout << "Ingrese un vector de n elementos\n";
    for (i = 0; i < f; i++)
        cin >> A[i];
}
int main() {
    int f;
    cout << "Ingrese el tamaño del vector: ";
    cin >> f;
    int *A = new int[f];
    leerVector(A, f);
    escribirVector(A, f);
}
```

## Matriz tamaño constante

```
#include <iostream>
using namespace std;
void escribirMatriz(int A[][2], int f, int c) {
    int i, j;
    cout << "Matriz ingresada\n";
    for (i = 0; i < f; i++) {
        for (j = 0; j < c; j++)
            cout << A[i][j] << "\t";
        cout << endl;
    }
}
void leerMatriz(int A[][2], int f, int c) {
    int i, j;
    cout << "Ingrese una matriz de 3 x 2\n";
    for (i = 0; i < f; i++)
        for (j = 0; j < c; j++)
            cin >> A[i][j];
}
int main() {
    const int F = 3;
    const int C = 2;
    int A[F][C];
    leerMatriz(A, F, C);
    escribirMatriz(A, F, C);
}
```

```
C:\Users\elpep\Documents\Sin Nombre2.exe
Ingrese una matriz de 3 x 2
1
2
3
4
5
6
Matriz ingresada
1      2
3      4
5      6
-----
Process exited after 4.129 seconds with
return value 0
Presione una tecla para continuar . . .
```

## Matriz tamaño variable

```
#include <iostream>
using namespace std;
void escribirMatriz(int *A[], int f, int c) {
    int i, j;
    cout << "Matriz ingresada\n";
    for (i = 0; i < f; i++)
    {
        for (j = 0; j < c; j++)
            cout << A[i][j] << "\t";
        cout << endl;
    }
}
void leerMatriz(int *A[], int f, int c) {
    int i, j;
    cout << "Ingrese una matriz de f x c\n";
    for (i = 0; i < f; i++)
        for (j = 0; j < c; j++)
            cin >> A[i][j];
}
int main() {
    int f, c, i, j;
    cout << "Ingrese la cantidad de filas de la matriz: ";
    cin >> f;
    cout << "Ingrese la cantidad de columnas de la matriz: ";
    cin >> c;
    int** A = new int* [f];
    for (i = 0; i < f; i++)
        A[i] = new int[c];
    leerMatriz(A, f, c);
    escribirMatriz(A, f, c);
}
```

```
C:\Users\elpep\Documents\Sin Nombre2.exe
Ingrese la cantidad de filas de la matriz: 3
Ingrese la cantidad de columnas de la matriz: 2
Ingrese una matriz de f x c
1
2
3
4
5
6
Matriz ingresada
1      2
3      4
5      6
-----
Process exited after 9.61 seconds with return value 0
Presione una tecla para continuar . . .
```

## Matriz tamaño variable

```
#include <iostream>
using namespace std;
void escribirMatriz(int **A, int f, int c) {
    int i, j;
    cout << "Matriz ingresada\n";
    for (i = 0; i < f; i++)
    {
        for (j = 0; j < c; j++)
            cout << A[i][j] << "\t";
        cout << endl;
    }
}
void leerMatriz(int **A, int f, int c) {
    int i, j;
    cout << "Ingrese una matriz de f x c\n";
    for (i = 0; i < f; i++)
        for (j = 0; j < c; j++)
            cin >> A[i][j];
    // *(A[i]+j) o *((*A+i)+j) o *(&A[0][0]+ c*i+j) o *(A+c*i+j)
}
int main() {
    int f, c, i, j;
    cout << "Ingrese la cantidad de filas de la matriz: ";
    cin >> f;
    cout << "Ingrese la cantidad de columnas de la matriz: ";
    cin >> c;
    int** A = new int* [f];
    for (i = 0; i < f; i++)
        A[i] = new int[c];
    leerMatriz(A, f, c);
    escribirMatriz(A, f, c);
}
```

```
C:\Users\elpep\Documents\Sin Nombre2.exe
Ingrese la cantidad de filas de la matriz: 3
Ingrese la cantidad de columnas de la matriz: 2
Ingrese una matriz de f x c
1
2
3
4
5
6
Matriz ingresada
1      2
3      4
5      6
-----
Process exited after 9.61 seconds with return value 0
Presione una tecla para continuar . . .
```

## Ejercicio 1

2. [5.0 ptos.] Utilice un apuntador para crear la siguiente matriz  $A$  de  $N \times N$  ( $N \leq 10$ ), en donde cada elemento se determina como la multiplicación del número de fila por 10 y agregando el número de columna.

### Ejemplo de salida para N=6:

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

A continuación, realice las siguientes operaciones:

- Extraer la segunda columna de  $A$  usando un apuntador. Mostrar como vector columna.
- Extraer la segunda fila de  $A$  usando un apuntador. Mostrar como vector fila.
- Usando un apuntador extraer la submatriz que se encuentra en la mitad superior de la diagonal principal.
- Usando un apuntador, extraer la submatriz que se encuentra en la mitad inferior de la diagonal secundaria.

## Ejercicio 1 Salida

```
n= 5
segunda columna
1
11
21
31
41
segunda fila
10 11 12 13 14

c)
0 1 2 3 4
0 0 12 13 14
0 0 0 23 24
0 0 0 0 34
0 0 0 0 0

d)
0 0 0 0 0
0 0 0 0 14
0 0 0 23 24
0 0 32 33 34
0 41 42 43 44

-----
Process exited after 1.464 seconds with
return value 0
Presione una tecla para continuar . . .
```

## Ejercicio 2

3. [3.0 ptos.] Escriba en C++ usando apuntadores, la función *productoMatrices()* que dado dos matrices cuadradas de orden n, genere la tercera. Escriba las funciones que crea conveniente.

## Ejercicio 2 Salida

```
Ingrese el orden de la matriz cuadrada: 5
Valores de los elementos de la matrices:
mat1:
4 6 3 5 1
1 10 7 4 2
2 10 5 3 7
4 7 2 2 5
5 1 3 1 2
mat2:
9 1 6 5 10
5 4 6 4 2
9 8 1 4 1
2 8 9 7 4
9 1 1 3 1

mat1 x mat2:
112 93 109 94 76
148 131 111 107 55
182 113 111 112 64
138 69 91 85 69
97 43 50 54 61

-----
Process exited after 7.202 seconds with return
value 0
```

## Ejercicio 3

4. [6.0 ptos.] El bit de paridad es un mecanismo simple para detectar errores en los datos trasmítidos por canales poco seguros como las líneas de teléfono. La idea básica es trasmítir un bit adicional después de cada grupo de 8 bits de tal manera que un error en un solo bit en la trasmisión pueda ser detectado. Los bits de paridad pueden ser determinados tanto para paridad par como paridad impar. Si se selecciona paridad par entonces el bit de paridad que se trasmítirá se escoge de tal manera que la cantidad total de unos (1's) trasmítidos (8 bits de datos más el bit de paridad) sea par. Si se selecciona paridad impar, el bit de paridad se escoge de tal manera que la cantidad total de unos (1's) trasmítidos sea impar. Escriba un programa que determine el bit de paridad para grupos de 8 bits ingresados por el usuario usando paridad par. Su programa en C++ usando apuntadores debe leer cadenas de 8 bits hasta que el usuario ingrese una línea en blanco. Después que se ingrese cada cadena válida, el programa debe de mostrar un mensaje indicando si el bit de paridad es 1 o 0. Si la cadena no es válida, debe mostrar un mensaje de error apropiado y solicitar el ingreso de una nueva cadena.

## Ejercicio 3 Salida

```
Ingrese una cadena de bits(Enter para terminar): 10101010
Error!
La cadena no tiene 8 bits
Ingrese una cadena de bits: hola
Error!
La cadena no tiene 8 bits
La cadena no es binaria!!
Ingrese una cadena de bits: 11010101
Bit de paridad= 1
Ingrese una cadena de bits(Enter para terminar): 10101010
Bit de paridad= 0
Ingrese una cadena de bits(Enter para terminar): 11111111
Bit de paridad= 0
Ingrese una cadena de bits(Enter para terminar): 00000000
Bit de paridad= 0
Ingrese una cadena de bits(Enter para terminar):

-----
Process exited after 46.18 seconds with return value 0
Presione una tecla para continuar . . .
```

**FIN**



# FUNDAMENTOS DE PROGRAMACIÓN

## Punteros y Cadenas (Parte 1)

Profesor: Carlos Diaz

cdiazd@uni.edu.pe

### Punteros y Cadenas

- Ivalues y rvalues
- Comparación entre vector de enteros y caracteres
- Cadenas
- Cadenas estilo C
- Importancia del carácter nulo '\0'
- Completando una vector de caracteres
- Declaración de cadenas C
- Lectura de cadenas
- Sintaxis de getline()
- Algunas funciones de la librería cstring
- Arrays de string

### Ivalues y rvalues

- Los nombres de las variables son **Ivalues** ("valores a la izquierda"), ya que pueden usarse del lado izquierdo de un operador de asignación. Son el destino de un valor.
- Las constantes son **rvalues** ("valores a la derecha"), ya que solo se pueden usar del lado derecho de un operador de asignación. Es la fuente de un valor.  
`x = 3; // x es lvalue y 3 es rvalue`
- Observe que los **Ivalues** también se pueden usar como rvalues, pero no al revés.  
`3 = x; // no es valido`
- En una tarea compuesta, la variable actúa tanto como un **lvalue** como un **rvalue**.  
`x = x + 3; // x es lvalue y rvalue a la vez y 3 es rvalue`
- Es conclusión, una entidad que puede ser destino de un valor se llama **lvalue** y una entidad que puede ser la fuente de un valor es un **rvalue**.

## Comparación entre vector de enteros y caracteres

- En un **vector de enteros**, el nombre del vector es una **variable puntero** que apunta al primer elemento del vector, es decir contiene la dirección del primer elemento del vector. Es un **lvalue**.
- En un **vector de caracteres**, el nombre del vector no es una variable puntero, define un valor de puntero al primer carácter del vector, es decir es la dirección del primer carácter y devuelve la cadena completa del vector. Es un **rvalue**.

### Ejemplo 1:

```
#include<iostream>
//Permite fijar configuraciones regionales
#include<locale>
using namespace std;
int main()
{
    setlocale(LC_ALL, ""); //Configuración de la PC actual
    int v[5] = { 1,2,3,4,5 }; //5 enteros
    char c[5] = { 'a','b','c','d','e' }; //5 caracteres
    cout << "Dirección 1ra celda v: " << v << endl;
    cout << "Dirección 2da celda v+1: " << v + 1 << endl;
    cout << "Dirección 3ra celda v+2: " << v + 2 << endl;
    cout << "Cadena a partir 1ra celda c: " << c << endl;
    cout << "Cadena a partir 2da celda c+1: " << c + 1 << endl;
    cout << "Cadena a partir 3ra celda c+2: " << c + 2 << endl;
    return 0;
}
```

The screenshot shows the Microsoft Visual Studio Debug Console window. It displays memory dump information for variables v and c. The memory dump for v shows addresses 00000746C0FF568, 00000746C0FF56C, and 00000746C0FF570, with their corresponding values 1, 2, and 3. The memory dump for c shows addresses 00000746C0FF574, 00000746C0FF578, 00000746C0FF57C, 00000746C0FF580, and 00000746C0FF584, with their corresponding values 'a', 'b', 'c', 'd', and 'e'. Below the dump, a message in Spanish encourages closing the console when debugging is finished.

## Comparación entre vector de enteros y caracteres

### Ejemplo 2:

```
#include<iostream>
using namespace std;
int main()
{
    int v[5] = { 1,2,3,4,5 }; //5 enteros
    char c[5] = { 'a','b','c','d','e' }; //5 caracteres
    cout << "Contenido 1ra celda *v: " << *v << endl;
    cout << "Contenido 2da celda *(v+1): " << *(v + 1) << endl;
    cout << "Contenido 3ra celda *(v+2): " << *(v + 2) << endl;
    cout << "Contenido 1ra celda *c: " << *c << endl;
    cout << "Contenido 2da celda *(c+1): " << *(c + 1) << endl;
    cout << "Contenido 3ra celda *(c+2): " << *(c + 2) << endl;
    cout << "Contenido 1ra celda c[0]: " << c[0] << endl;
    cout << "Contenido 2da celda c[1]: " << c[1] << endl;
    cout << "Contenido 3ra celda c[2]: " << c[2] << endl;
    cout << "Cadena a partir celda &c[0]: " << &c[0] << endl;
    cout << "Cadena a partir celda &c[1]: " << &c[1] << endl;
    cout << "Cadena a partir celda &c[2]: " << &c[2] << endl;
    return 0;
}
```

The screenshot shows the Microsoft Visual Studio Debug Console window. It displays the output of the code from Example 2. The output shows the contents of the integer array v and the character array c at their respective addresses, followed by the strings they represent. The console window also contains a message in Spanish about how to close it.

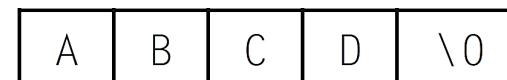
## Cadenas

- Aunque el lenguaje C++ contiene el tipo de clase de cadena C++ (**<cstring>**), vamos a estudiar la cadena C (o la cadena estilo C) por dos razones.

- Hay algunos programas (incluidas algunas bibliotecas C++) que todavía usan cadenas C.
- La clase de cadena C++ usa algunas cadenas C en sus definiciones, por lo tanto, es necesario un conocimiento básico de las cadenas C para comprender las cadenas C ++.

## Cadena estilo C

- Una cadena estilo C es un tipo de dato compuesto por un array de caracteres (**char**), terminado por un carácter nulo (**'\0'**).
- Por ejemplo, la cadena "ABCD", en memoria consta de cinco caracteres: 'A', 'B', 'C', 'D' y '\0', es decir, la cadena "ABCD" es un array de cinco elementos de tipo char.



- El número total de caracteres del array es siempre igual a la longitud de la cadena más 1.

## Importancia del carácter nulo '\0'

- El carácter nulo indica que allí termina la cadena.

¿Cuál es la diferencia entre estas dos declaraciones?

```
char c1[5] = { 'a','b','c','d','e' };
```

```
char c2[5] = { 'a','b','c','d','\0' };
```

Ejemplo 3:

```
#include<iostream>
using namespace std;
```

```
int main()
```

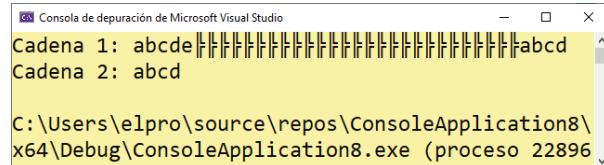
```
{
```

```
    char c1[5] = { 'a','b','c','d','e' };
    char c2[5] = { 'a','b','c','d','\0' };
```

```
    cout << "Cadena 1: " << c1 << endl; //No sabe donde detenerse
```

```
    cout << "Cadena 2: " << c2 << endl; //Termina con '\0'
```

```
    return 0;
}
```



## Completando un vector de caracteres

- Para leer un carácter usamos la función `cin.get()`. La llamada `cin.get(car)` copia el carácter en la variable `car` y devuelve `1`.

Ejemplo 4: Sin carácter nulo.

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    char c[25];
```

```
    int i = 0;
```

```
    cout << "Escriba una cadena : ";
```

```
    do
```

```
    {
```

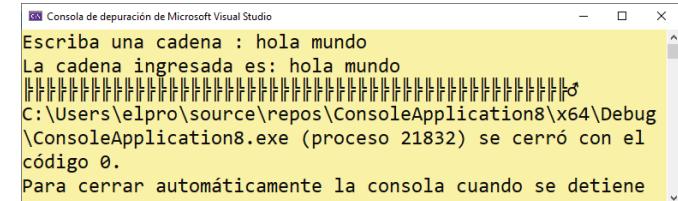
```
        cin.get(c[i]);
```

```
        i++;
    }
```

```
    } while (c[i - 1] != '\n');
```

```
    cout << "La cadena ingresada es: " << c << endl;
```

```
    return 0;
}
```



## Completando un vector de caracteres

Ejemplo 5: Con carácter nulo.

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    char c[25];
```

```
    int i = 0;
```

```
    cout << "Escriba una cadena : ";
```

```
    do
```

```
    {
```

```
        cin.get(c[i]);
```

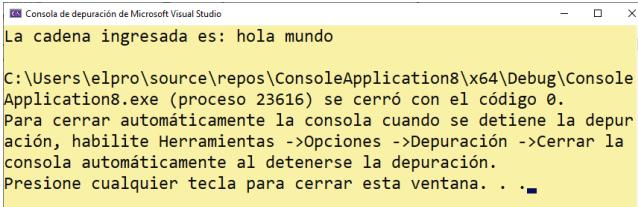
```
        i++;
    }
```

```
    } while (c[i - 1] != '\n');
```

```
    c[i-1] = '\0'; //Sobrescribimos '\n'
```

```
    cout << "La cadena ingresada es: " << c << endl;
```

```
    return 0;
}
```



## Declaración de cadenas C

- Las cadenas se declaran como cualquier array.
- Las cadenas de caracteres pueden inicializarse en el momento de la declaración. El compilador añade automáticamente el carácter '\0'.
- Una cadena **no** se puede inicializar fuera de la declaración. Para asignar una cadena a otra hay que utilizar la función `strcpy()`. La función `strcpy()` copia los caracteres de la cadena fuente a la cadena destino.
- Para utilizar la función `strcpy()` debemos incluir la librería `<cstring>` que es la versión de librería estándar C ++ del archivo de librería estándar C `<string.h>`

# Declaración de cadenas C

## Ejemplo 6:

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
    char cadena1[5]; //Vector de caracteres. Maximo 5 caracteres
    char cadena2[5] = "hola"; //Maximo 5 caracteres incluido el caracter nulo
    char cadena3[5] = "hi"; //Maximo 5 caracteres incluido el caracter nulo
    char cadena4[] = "la luna"; //Adopta el tamaño de 8 caracteres incluido el caracter nulo
    cout << "Cadena 1: " << cadena1 << endl;
    cout << "Cadena 2: " << cadena2 << endl;
    cout << "Cadena 3: " << cadena3 << endl;
    cout << "Cadena 4: " << cadena4 << endl;
    cout << "Cantidad de caracteres cadena 3: " << strlen(cadena3) << endl; // 2
    cout << "Cantidad de celdas del vector cadena 3: " << sizeof(cadena3) << endl; // 5
    for (int i = 0; i < strlen(cadena4); i++)
        cout << "cadena4[" << i << "]:" << cadena4[i] << endl;
    strcpy(cadena1, "FIN");
    cout << "Cadena 1: " << cadena1 << endl;
    return 0;
}
```

# Lectura de cadenas C

- La lectura usual de una palabra se realiza con el objeto `cin` y el operador `>>`. Si se aplica a una cadena produce anomalías, ya que el objeto `cin` termina la operación de lectura cuando encuentra un espacio en blanco.
- Para leer cadenas se utiliza la función `getline()` en unión con `cin`.
- La función `getline()` permite a `cin` leer la cadena completa, incluyendo cualquier espacio en blanco.

# Lectura de cadenas C

## Ejemplo 7:

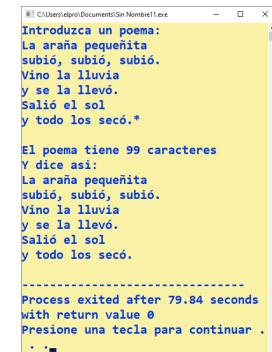
```
#include<iostream>
using namespace std;
int main()
{
    char palabra[10];
    char frase[10];
    cout << "Introduzca una palabra: ";
    cin >> palabra;
    cout << "Introduzca una frase: ";
    cin.ignore(); //Limpia el buffer que tiene el salto de linea del cin anterior
    cin.getline(frase, sizeof(frase));
    cout << "La palabra: " << palabra << endl;
    cout << "La frase: " << frase << endl;
    return 0;
}
```

# Sintaxis de getline()

- En su forma más general, el método `getline()` tiene la sintaxis:  
`cin.getline(Cadena,TamañoIncluidoElNulo,caracterDeTerminación);`

## Ejemplo 8: Escriba un poema que termine con \*

```
#include<iostream>
#include <cstring>
using namespace std;
int main()
{
    char poema[150];
    cout << "Introduzca un poema:\n";
    cin.getline(poema, sizeof(poema), '*');
    cout << "\nEl poema tiene " << strlen(poema) << " caracteres" << endl;
    cout << "Y dice asi:" << endl;
    cout << poema << endl;
    return 0;
}
```



# Algunas funciones de la librería cstring

Función	Descripción
strlen (str);	Esta función devuelve la longitud de la cadena str. La longitud de la cadena es el número de caracteres en la cadena sin el carácter de terminación '\0'
strcpy (dest, ori);	Esta es una función de copia de cadena. El primer parámetro es la cadena destino y el segundo parámetro es la cadena de origen. La función strcpy() copiará el contenido de la cadena de origen al destino, incluido el carácter nulo de terminación '\0'
strcat (dest, ori);	Esta es una función de concatenación de cadenas. El primer parámetro es la cadena destino y el segundo parámetro es la cadena de origen. La función strcat() concatenará/agregará el contenido de origen al destino. El carácter nulo de terminación de destino se sobrescribe con el primer carácter de origen y se introduce un carácter nulo en el destino al final de la nueva cadena.
strcmp (str1, str2);	Esta función compara dos cadenas. El primer parámetro es str1 y el segundo parámetro es str2 que será comparado por esta función. Comienza comparando el primer carácter de ambas cadenas, compara hasta que encuentra un carácter que no coincide o el nulo. Esta función devuelve un valor entero. El 0 se devuelve cuando las cadenas son iguales. Se devuelve un valor negativo cuando el carácter encontrado tiene un valor más bajo en str1 que en str2 (str1<str2). Devuelve un positivo cuando el carácter no coincidente encontrado tiene un mayor valor en str1 que str2 (str1>str2).

# Algunas funciones de la librería cstring

## Ejemplo 9:

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char cadena1[100];
    char cadena2[100] = "No hay camino para la verdad, la verdad es el camino.";
    char cadena3[100] = " Mahatma Gandhi.";
    char cadena4[100] = " Mohandas Karamchand Gandhi.";
    // Longitud de la cadena 1
    cout << "strlen(cadena1) : " << strlen(cadena1) << endl;
    // Longitud de la cadena 2
    cout << "strlen(cadena2) : " << strlen(cadena2) << endl;
    // Copiando cadena 2 en cadena 1
    strcpy(cadena1, cadena2);
    cout << "strcpy( cadena1, cadena2) : " << cadena1 << endl;
    // Concatenado cadena 1 con cadena 3
    strcat(cadena1, cadena3);
    cout << "strcat( cadena1, cadena3) : " << cadena1 << endl;
    // Comparando cadena 3 con cadena 4
    cout << "strcmp( cadena3, cadena4) : " << strcmp(cadena3, cadena4) << endl;
    // Comparando cadena 3 con cadena 2
    cout << "strcmp( cadena4, cadena3) : " << strcmp(cadena4, cadena3) << endl;
    return 0;
}
```

# Algunas funciones de la librería cstring

Función	Descripción
strchr(str, ch);	Esta función devuelve un puntero a la primera aparición del carácter ch en la cadena str. Esta función se utiliza para localizar la primera aparición de un carácter en la cadena.
strrchr(str, ch);	Esta función devuelve un puntero a la última aparición del carácter ch en la cadena str. Esta función se utiliza para localizar la última aparición de un carácter en la cadena.
strstr(str1, str2);	Esta función se utiliza para localizar una subcadena y devuelve un puntero a la primera aparición de la cadena str2 en la cadena str1 o un puntero nulo si str2 no forma parte de str1.
strncpy(str1, str2, n)	Esta función copia de la cadena str2 a la cadena str1 los n primeros caracteres indicados. Este n no puede ser superior al tamaño de str1.
strncmp(s1, s2, n)	Esta función es lo mismo que la función strcmp, excepto que sólo se compara los n primeros caracteres indicados.
strncat (dest, ori, n);	Esta función añade los primeros n caracteres de str2 a str1

# Algunas funciones de la librería cstring

## Ejemplo 10:

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char cadena1[100] = "";
    char cadena2[100] = "No hay camino para la verdad, la verdad es el camino.";
    char cadena3[100] = " Mahatma Gandhi.";
    // Puntero al carácter 'c' de cadena 2
    char* p1 = strchr(cadena2, 'c');
    cout << "strchr( cadena2, 'c') : " << p1 << endl;
    // Puntero al último carácter 'c' de cadena 2
    char* p2 = strrchr(cadena2, 'c');
    cout << "strrchr( cadena2, 'c') : " << p2 << endl;
    // Puntero a la cadena "verdad" de cadena 2
    char* p3 = strstr(cadena2, "verdad");
    cout << "strstr( cadena2, \"verdad\") : " << p3 << endl;
    // Copiando 13 caracteres de cadena 2 en cadena 1
    strncpy(cadena1, cadena2, 13);
    cout << "strncpy( cadena1, cadena2, 13) : " << cadena1 << endl;
    // Comparando los 5 primeros caracteres cadena 1 y cadena 2
    cout << "strcmp( cadena1, cadena2, 5) : " << strcmp(cadena1, cadena2, 5) << endl;
    // Concatenado los 8 primeros caracteres de cadena 3 en cadena 1
    strncat(cadena1, cadena3, 8);
    cout << "strncat( cadena1, cadena3, 8) : " << cadena1 << endl;
    return 0;
}
```

## Ejercicio 1

- Escriba un programa que lea una frase y nos diga cuantas vocales mayúsculas o minúsculas tiene, el programa termina al ingresar **CTRL+Z** o **^Z**.

**Sugerencia:** Utilice el método `toupper(car)` o `tolower(car)` para convertir a mayúsculas o minúsculas respectivamente.

**Nota:** EOF (End Of File) es un parámetro booleano útil para facilitar el cierre de bucles de extracción de datos desde archivo. En C++, EOF es una constante de tipo entero (normalmente -1) que es el retorno que envían distintas funciones de extracción de información desde archivos al llegar a un final de archivo y no existir más datos. También se puede "simular" EOF mediante una entrada de teclado, normalmente **CTRL+Z**

## Ejercicio 2

- Escriba un programa lee un texto y lo visualiza, escribiendo todas las vocales en minúscula y las consonantes en mayúsculas.

**Sugerencia:** Tome en cuenta las siguientes funciones.

`isalpha(car)` evalúa si car es una letra.

`isalnum(car)` evalúa si car es una letra o dígito.

`isupper(car)` evalúa si car es una letra mayúscula.

`islower(car)` evalúa si car es una letra minúscula.

`isdigit(car)` evalúa si car es un dígito.

`isspace(car)` evalúa si car es un espacio.

`ispunct(car)` evalúa si car es un carácter de puntuación.

## Array de strings

//Array de palabras sin espacio

```
#include <iostream>
using namespace std;
int main()
{
    char nombres[10][40];
    int i = 0;
    cout << "Ingrese 5 nombres: " << endl;
    for (i = 0; i < 5; i++)
        cin >> nombres[i];
    cout << endl << "Los nombres ingresados son :" << endl;
    for (i = 0; i < 5; i++)
        cout << nombres[i] << endl;
    return 0;
}
```

## Array de strings

//Array de frases con espacio

```
#include <iostream>
using namespace std;
int main()
{
    char frases[10][40];
    int i = 0;
    cout << "Ingrese 5 frases: " << endl;
    for (i = 0; i < 5; i++)
        cin.getline(frases[i], sizeof(frases[0]));
    cout << endl << "Las frases ingresados son :" << endl;
    for (i = 0; i < 5; i++)
        cout << frases[i] << endl;
    //Nota: Cantidad de filas y columnas del array
    cout << "Cantidad de filas: " << sizeof(frases) / sizeof(frases[0]) << endl;
    cout << "Cantidad de columnas: " << sizeof(frases[0]) << endl;
}

```

## Ejercicio 3

- Leer un texto cuyo máximo número de líneas sea 10 de una longitud máxima de 15 caracteres por línea, y escribir el mismo texto, pero intercambiando entre sí las líneas de mayor con la de menor longitud, es decir en orden ascendente.
- La lectura termina cuando se han leído las 10 líneas, o bien cuando se introduce una línea vacía.

## Funciones de conversión cstring

- La función `atoi(cadena)` convierte una cadena a un valor entero. La cadena debe tener la representación de un valor entero. Si la cadena no se puede convertir, `atoi( )` devuelve cero.
- La función `atof(cadena)` convierte una cadena a un valor de coma flotante. La conversión termina cuando se encuentre un carácter no reconocido. La cadena de caracteres debe tener una representación de caracteres de un número de coma flotante.

## Tokens

- La función `strtok(s1,s2)` analiza la cadena s1 en tokens (elemento más pequeño) delimitados por los caracteres encontrados en la cadena s2. Después de la llamada inicial `strtok(s1, s2)`, cada llamada sucesiva a `strtok(NULL,s2)` devuelve un puntero al siguiente token encontrado en s1.

Ejemplo:

```
int main()
{
    char frase[100], *token;
    int i = 0;
    cout << "Escriba una cadena : ";
    cin.getline(frase, sizeof(frase));
    token = strtok(frase, " /:");
    cout << token << endl;
    while (token) // (true)
    {
        token = strtok(NULL, " /:");
        if (token != NULL)
            cout << token << endl;
        else
            //break;
    }
    return 0;
}
```

## Ejercicio 4

- Escriba un programa que lea una cadena y sume todos los números que aparecen en ella. Muestre la suma.

FIN



# FUNDAMENTOS DE PROGRAMACIÓN

## Estructuras

Profesor: Carlos Diaz

cdiazd@uni.edu.pe

### Clase 10: Estructura de datos

- Estructura, declaración y definición
- Estructuras anidadas
- Array de estructuras
- Paso de estructuras a funciones
- Punteros a estructuras

### Estructura

- Una estructura es un tipo de dato definido por el usuario, que se debe declarar antes de que se pueda utilizar. Es una colección de uno o más tipos de elementos denominados miembros, cada uno de los cuales puede ser un tipo de dato diferente.
- Declaración de una estructura:
- ```
struct <nombre de la estructura>
{
    <tipo de dato miembro1><nombre miembro1>
    <tipo de dato miembro2><nombre miembro2>
    ...
    <tipo de dato miembroN><nombre miembroN>
};
```

## Ejemplo 1

- Declaración, definición y asignación de una estructura llamada persona.

```
#include <iostream>
#include<string>
using namespace std;
struct persona { // nombre de la estructura
    string nombre;
    int nota;
} p1, p2 = { "Juan",20 };
int main() {
    persona p3;
    float prom;
    p1.nombre = "Maria";
    p1.nota = 18;
    p3 = p1;
    cout << "Nombre de p1 " << p1.nombre << endl;
    cout << "Nombre de p2 " << p2.nombre << endl;
    cout << "Nombre de p3 " << p3.nombre << endl;
    prom = (p1.nota + p2.nota + p3.nota) / 3.0;
    cout << "El promedio es: " << prom;
    return 0;
}
```

## Ejemplo 2

- Una estructura anidada es una estructura contiene otras estructuras.
- 1. Se creara una estructura DATOS que contenga los miembros Nombre y Edad.

- 2. Se creara la estructura ALUMNO que contenga la estructura DATOS además de los miembros Curso y Notas (array de 3 enteros)

```
#include <iostream>
#include<string>
using namespace std;
struct datos
{
    string nombre;
    int edad;
};
struct alumno {
    datos dat;
    string curso;
    int notas[3];
};
```

## Ejemplo 2

```
int main() {
    //Introduciendo datos
    alumno estudiante;
    int i, s;
    double promedio;
    cout << "Escriba el nombre: ";
    getline(cin, estudiante.dat.nombre);
    cout << "Escriba la edad: ";
    cin >> estudiante.dat.edad;
    cin.ignore();
    cout << "Escriba el curso: ";
    getline(cin, estudiante.curso);
    cout << "Ingrese las notas\n";
    for (i = 0; i < sizeof(estudiante.notas) / sizeof(estudiante.notas[0]); i++) {
        cout << "Nota " << i + 1 << ": ";
        cin >> estudiante.notas[i];
    }
    //Mostrando promedio
    s = 0;
    for (i = 0; i < sizeof(estudiante.notas) / sizeof(estudiante.notas[0]); i++)
        s = s + estudiante.notas[i];
    promedio = s / 3.0;
    cout << "El alumno " << estudiante.dat.nombre << ", tiene promedio: " << promedio;
}
```

## Array de estructuras

- Ejemplo 3: Modifique el main() del ejemplo anterior para leer los datos de 3 estudiantes.

```
int main() {
    //Introduciendo datos
    alumno estudiante[3];
    int i, j;
    for (i = 0; i < sizeof(estudiante) / sizeof(estudiante[0]); i++) {
        cout << "Escriba el nombre " << i + 1 << ": ";
        getline(cin,estudiante[i].dat.nombre);
        cout << "Escriba el curso " << i + 1 << ": ";
        getline(cin,estudiante[i].curso);
        cout << "Escriba la edad " << i + 1 << ": ";
        cin >> estudiante[i].dat.edad;
        cout << "Ingrese las notas " << i + 1 << ": " << endl;
        for (j = 0; j < sizeof(estudiante[i].notas)/sizeof(estudiante[i].notas[0]); j++) {
            cout << "Nota " << j + 1 << ": ";
            cin >> estudiante[i].notas[j];
        }
        cin.ignore();
    }
}
```

## Paso de estructuras a funciones

- C++ permite pasar estructuras a funciones, bien por valor o bien por referencia, utilizando el operador &.
- Si la estructura es grande, el tiempo necesario para copiar un parámetro struct a la pila puede ser prohibitivo. En tales casos, se debe considerar el método de pasar la dirección de la estructura.
- **Ejemplo 4:** Modifique el ejemplo anterior para que la lectura de datos este en una función. Y también cree otra función que muestre los datos ingresados.

```
#include <iostream>
#include<string>
using namespace std;
struct datos
{
    string nombre;
    int edad;
};
struct alumno {
    datos dat;
    string curso;
    int notas[3];
};
```

## Ejemplo 4

```
void leerDatos(alumno& pupilo) {
    int i;
    cout << "Escriba el nombre: ";
    getline(cin,pupilo.dat.nombre);
    cout << "Escriba el curso: ";
    getline(cin,pupilo.curso);
    cout << "Escriba la edad: ";
    cin >> pupilo.dat.edad;
    cout << "Ingrese las notas:" << endl;
    for (i = 0; i < 3; i++) {
        cout << "Nota " << i + 1 << ": ";
        cin >> pupilo.notas[i];
    }
}
void mostrarDatos(alumno& pupilo) {
    int i;
    cout << "Nombre: " << pupilo.dat.nombre << endl;
    cout << "Curso: " << pupilo.curso << endl;
    cout << "Edad: " << pupilo.dat.edad << endl;
    cout << "Las notas: " << endl;
    for (i = 0; i < 3; i++) {
        cout << "Nota " << i + 1 << ": ";
        cout << pupilo.notas[i] << endl;
    }
}
```

## Ejemplo 4

```
int main() {
    //Introduciendo datos
    alumno estudiante[3];
    int i, n;
    n = sizeof(estudiante) / sizeof(estudiante[0]);
    cout << "Introduciendo datos\n";
    for (i = 0; i < n; i++) {
        cout << "Estudiante " << i + 1 << endl;
        leerDatos(estudiante[i]);
        cin.ignore();
    }
    //Mostrando datos
    cout << "Mostrando datos\n";
    for (i = 0; i < n; i++) {
        cout << "Estudiante " << i + 1 << endl;
        mostrarDatos(estudiante[i]);
    }
}
```

## Ejercicio 1

- Modifique el ejercicio 4 para que la cantidad de estudiantes se lea desde el teclado durante la ejecución.

## Punteros a estructuras

- Se puede declarar un puntero a una estructura tal como se declara un puntero a cualquier otro objeto.
- Cuando se referencia un miembro de una estructura usando el nombre de la estructura, se emplea el operador punto (.).
- En cambio, cuando se referencia una estructura utilizando el puntero estructura, se emplea el operador flecha (->) para acceder a un miembro de ella.
- **Ejemplo 5:** Leer y mostrar el código de un producto.

```
#include <iostream>
#include <string>
using namespace std;
struct articulo {
    string nombre;
    int codigo;
};
```

## Ejemplo 5

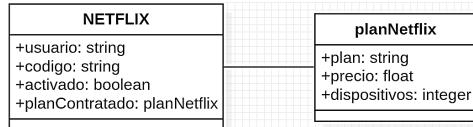
```
void leerarticulo(articulo* pArticulo) {
    cout << "Nombre del articulo: ";
    getline(cin, pArticulo->nombre);
    cout << "Codigo del articulo: ";
    cin >> pArticulo->codigo;
}
void mostrararticulo(articulo* Articulo) {
    cout << "Nombre del articulo: " << Articulo->nombre << endl;
    cout << "Codigo del articulo: " << Articulo->codigo << endl;
}
int main() {
    articulo* pArt = new articulo;
    leerarticulo(pArt);
    mostrararticulo(pArt);
    return 0;
}
```

## Ejercicio 2

- Netflix tiene los siguientes planes para su servicio de streaming.

| Plan                 | Básico | Estándar | Premium |
|----------------------|--------|----------|---------|
| Precio               | 24.9   | 34.9     | 44.9    |
| Dispositivos máximos | 1      | 2        | 4       |

- Escriba un programa que solicite la cantidad de clientes a introducir a una base de datos, según la siguiente estructura.



- El campo usuario almacena el nombre y apellido del cliente.
- El campo código es una serie de números y letras.
- El campo activado significa si el usuario está activo o no. Al principio su valor es True.
- El campo planContratado es una estructura.
- El plan puede ser Básico, Estándar o Premium.
- El precio puede ser 24.9, 34.9 o 44.9 (Esto se asigna automáticamente según el plan)
- Los dispositivos pueden variar desde 1 hasta el máximo según el plan. Si coloca una cantidad incorrecta el programa solicita un nuevo valor.

## Ejercicio 2

- El programa, luego de completar la base de datos, muestra el siguiente menú:

1. Informe de todos los clientes. Ordenados como primera prioridad por plan (Básico, Estándar o Premium) y segunda prioridad el usuario. Los usuarios desactivados no se muestran.
2. Desactive a usuario mediante su código. Colocando False en el campo activado.
3. Ingresar un nuevo usuario

**Nota:** El informe muestra todos los campos (usuario, código, activado, plan, precio y dispositivos):

# FUNDAMENTOS DE PROGRAMACIÓN

Fin Clase 10

Profesor: Carlos Díaz



# FUNDAMENTOS DE PROGRAMACIÓN

POO

Profesor: Carlos Diaz

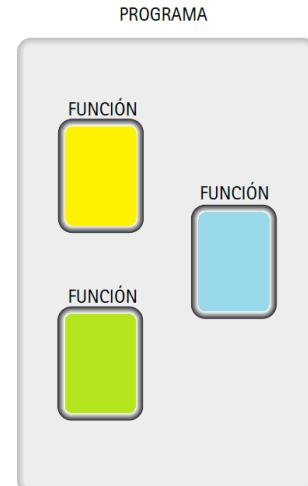
cdiazd@uni.edu.pe

## Programación Orientada a Objetos

- Paradigma de la programación estructurada
- Paradigma de la programación orientada a objetos
- POO
- Clases
- Objetos
- Herencia

## Paradigma de la programación estructurada

- Hasta ahora hemos usado la descomposición funcional (también llamada diseño estructurado), en la cual descomponemos un problema en módulos, donde cada módulo es una colección independiente de pasos que resuelve una parte del problema general.
- El proceso de aplicar una descomposición funcional se denomina **programación estructurada** (o procedural). Algunos módulos se traducen directamente a unas cuantas instrucciones del lenguaje de programación, mientras que otros son codificados como funciones con o sin argumentos.
- El resultado final es un programa que es una colección de funciones que interactúan entre si.

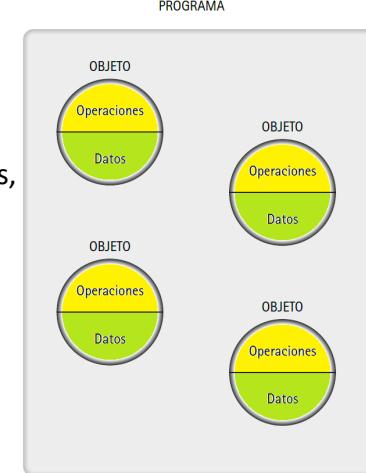


## Paradigma de la programación estructurada

- El diseño estructurado es satisfactorio para la programación en lo pequeño, pero con frecuencia no se deja expandir lo suficiente para programar situaciones más grandes.
- En la elaboración de grandes sistemas de software, el diseño estructurado tiene dos limitaciones importantes:
  1. En primer lugar, **produce una estructura inflexible**. Si el algoritmo de nivel superior requiere una modificación, los cambios podrían obligar también a una modificación de muchos algoritmos de nivel inferior.
  2. En segundo lugar, **no se presta fácilmente para la reutilización de código**. Es decir, la capacidad de usar partes de código —como son, o levemente modificadas— en otras secciones del programa o en otros programas. Es raro que se pueda tomar una función complicada y reutilizarla fácilmente en un contexto diferente.

## Paradigma de la programación orientada a objetos

- Una metodología que a menudo funciona mejor para crear grandes sistemas de software es el diseño orientado a objetos (DOO).
- La DOO se basa en el hecho de que un problema se puede dividir, no en tareas (calcular impuestos, imprimir un reporte), sino en modelar objetos del mundo real (cliente, factura).
- El DOO descompone un problema en objetos, o sea entidades independientes, que tienen datos y operaciones con esos datos.
- El proceso de aplicar un diseño orientado a objetos se denomina **programación orientada a objetos** (POO). El resultado final es un programa que es una colección de objetos que interactúan entre si.
- El DOO tiende a resultar en programas que son más flexibles y favorables para la reutilización de código que los programas producidos por medio del diseño estructurado.

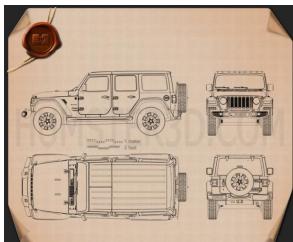


## Ejemplo de Clase y Objeto

- Un **objeto** es una estructura de datos compleja y flexible, que permite almacenar tanto información como la forma de operar con ella.
- Los objetos se definen mediante una plantilla llamada **clase**. Decimos que un objeto pertenece a una clase determinada cuando se ha creado (instanciado) a partir de ella.

**Ejemplo:** Se puede decir que la clase es como un "plano" para crear objetos.

Clase



Objetos



## Ejemplo de Clase y Objeto

- En una clase podemos definir variables que describen las características del objeto (llamadas **atributos** o **propiedades**) y funciones que usaremos para trabajar con esas variables (llamadas **métodos**)
- Dos objetos que pertenezcan a la misma clase (es decir, que hayan sido instanciados a partir de ella) tendrán los mismos atributos, pero los valores no tienen porqué ser los mismos.



## Ejemplo C++

```
#include <iostream>
#include <cmath>
using namespace std;
class Punto
{
    public: // Acceso o Visibilidad
        float x, y; // Atributos
    Punto() // Método Constructor
    {
        x = 1;
        y = 2;
        cout << "Construyo P(" << x << "," << y << ")\n";
    }
    ~Punto() // Método Destructor
    {
        cout << "Destruyo P(" << x << "," << y << ")\n";
    }
    float distancia() // Método distancia
    {
        return sqrt(x * x + y * y);
    }
};
```

```
int main()
{
    Punto p;
    float d;
    d = p.distancia();
    cout << d << endl;
    p.x = 3;
    p.y = 4;
    d = p.distancia();
    cout << d << endl;
    return 0;
}
```

## Modelado e identificación de objetos

- Un objeto tiene un **estado**, un **comportamiento** y una **identidad**.
- **Estado:** Conjunto de **valores** de todos los **atributos** de un objeto en un instante de tiempo específico.
- **Comportamiento:** Conjunto de **operaciones** que se pueden realizar sobre un objeto. Las operaciones pueden ser de observación del estado interno del objeto, o bien de modificación de dicho estado. El estado de un objeto puede evolucionar en función de la aplicación de sus operaciones (**métodos**). Estas operaciones se realizan tras la recepción de un mensaje externo enviado por otro objeto (**llamando a un método**).
- **Identidad:** Permite diferenciar los objetos, sin ambigüedad, independientemente de su estado. Es decir, es posible distinguir dos objetos en los cuales todos sus atributos sean iguales. Cada objeto posee su propia identidad de manera implícita. Cada objeto ocupa su propia posición en la memoria de la computadora.

## Ejemplo

- Tenemos la clase **Mascota**.

| Mascota         |
|-----------------|
| -nombre: string |
| -tipo: string   |

+actualizarDatos(): void

- Comportamiento del objeto **mascota1**.

| mascota1                 |
|--------------------------|
| +actualizarDatos(): void |

- Tres **objetos** diferentes **mascota1**, **mascota2** y **mascota3**, cada uno con su propia **Identidad**.

Los **objetos** **mascota1** y **mascota2** son dos gatos diferentes que tienen el mismo estado.

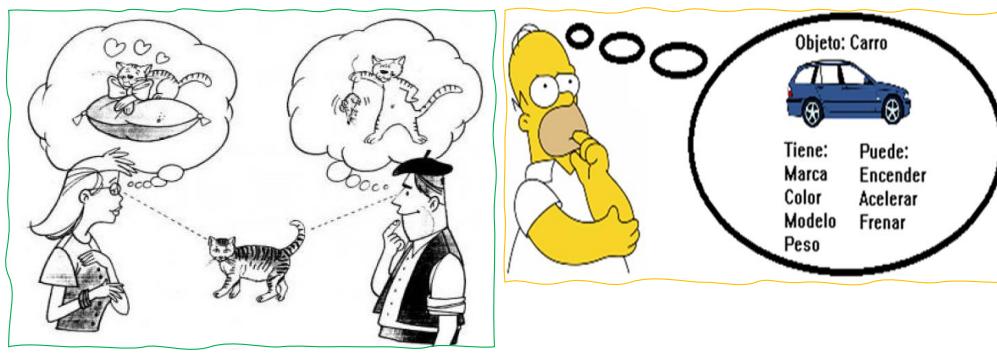
| mascota1         |
|------------------|
| -nombre: Micifuz |
| -tipo: gato      |
| -edad: 2         |
| -peso: 4.5       |

| mascota1         | mascota2         | mascota3          |
|------------------|------------------|-------------------|
| -nombre: Micifuz | -nombre: Micifuz | -nombre: Firulais |
| -tipo: gato      | -tipo: gato      | -tipo: perro      |
| -edad: 2         | -edad: 2         | -edad: 1          |
| -peso: 4.5       | -peso: 4.5       | -peso: 3          |

+actualizarDatos(): void

## Propiedades fundamentales de la orientación a objetos

- Los conceptos fundamentales de la programación orientada a objetos son: **abstracción**, **encapsulamiento** y **ocultación de datos**, **herencia (generalización)**, **reutilización** y **polimorfismo**.
- **Abstracción:** Es la propiedad que considera los aspectos más significativos o notables de un problema y expresa una solución en esos términos.



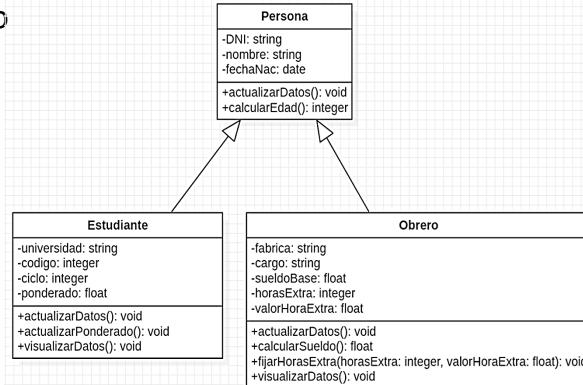
## Propiedades fundamentales de la orientación a objetos

- **Encapsulamiento:** Es la reunión en una cierta estructura de todos los elementos que a cierto nivel de abstracción se pueden considerar pertenecientes a una misma entidad.
- En este caso, los objetos que poseen las mismas características y comportamiento se agrupan en **clases** que son unidades de programación que **encapsulan** datos (**atributos**) y operaciones (**métodos**); la encapsulación oculta lo que hace un objeto de otros objetos del mundo exterior, por lo que se denomina también **ocultación de datos**.



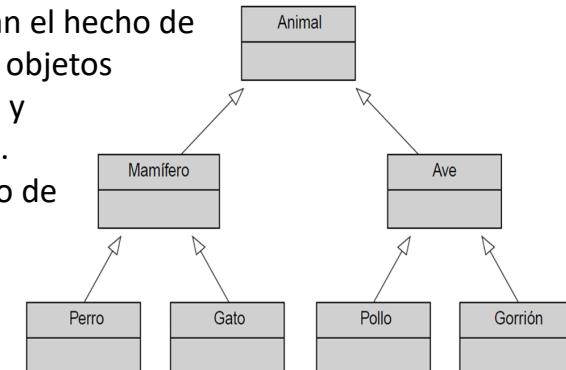
## Propiedades fundamentales de la orientación a objetos

- **Reutilización:** Este concepto significa que una vez que se ha creado, escrito y depurado una clase, se puede poner a disposición de otros programadores y ampliar sus capacidades.
- Una clase existente se puede ampliar añadiéndole nuevas características (atributos y métodos) mediante la **herencia**.
- Por ejemplo, las clases **Estudiante** y **Obrero** reutilizan las características de la clase **Persona** y le añaden nuevas características (atributos y métodos).



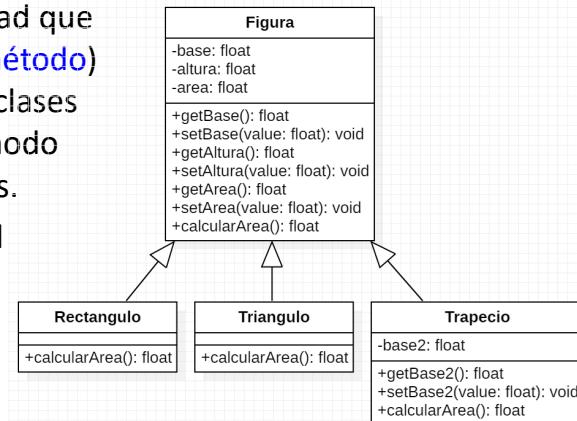
## Propiedades fundamentales de la orientación a objetos

- **Herencia:** Las clases modelan el hecho de que el mundo real contiene objetos con propiedades (**atributos**) y comportamiento (**métodos**). La **herencia** modela el hecho de que estos objetos tienden a organizarse en jerarquías. La clase de mayor jerarquía se llama **superclase** (clase base) y las clases de menor jerarquía se llaman **subclases** (clases derivadas).
- Cada subclase hereda las características de la superclase y además añade sus propias **características** (atributos y métodos).
- Una superclase puede a su vez ser también subclase de otras superclases.



## Propiedades fundamentales de la orientación a objetos

- **Polimorfismo:** Es la propiedad que permite a una operación (**método**) tener el mismo nombre en clases diferentes y que actúe de modo distinto en cada una de ellas.
- Así por ejemplo, tenemos el método **calcularArea()**, que calcula el área de diferente manera dependiendo de la superficie.



# Clases en C++

- Una **clase** es la descripción de un conjunto de objetos; consta de **métodos** (funciones miembro) y **atributos** (miembros dato) que resumen características comunes de un conjunto de objetos.
- Generalmente, una clase se puede definir como una descripción abstracta de un grupo de objetos, cada uno de los cuales se diferencia por un **estado** específico y es capaz de realizar una serie de **operaciones**.
- Una declaración de una clase consta de una palabra reservada **class** y el nombre de la clase. Se utilizan tres diferentes *especificadores de acceso* para controlar el acceso a los miembros de la clase. Estos accesos son: **public**, **private** y **protected**.
- Sintaxis:

```
class nombre_clase {
public:
    // miembros públicos
protected:
    // miembros protegidos
private:
    // miembros privados
};
```

## Ejemplo

```
#include <iostream>
#include <cmath>
using namespace std;
class Punto
{
private: //Acceso o Visibilidad
    float x, y;
public: //Acceso o Visibilidad
    Punto(x,y) //Constructor
    {
        this->x;
        this->y;
        cout << "Construyo P(" << x << "," << y << ")\n";
    }
    ~Punto() //Destructor
    {
        cout << "Destruyo P(" << x << "," << y << ")\n";
    }
    float getX() { return x; }
    void setX(float x) { this->x = x; }
    float getY() { return y; }
    void setY(float y) { this->y = y; }
    float distancia()
    {
        return sqrt(x * x + y * y);
    }
};
```

```
int main()
{
    Punto p(3,4);
    float d, x, y;
    d = p.distancia();
    cout << d << endl;
    cout << "Ingrese coordenada x: ";
    cin >> x;
    p.setX(x);
    cout << "Ingrese coordenada y: ";
    cin >> y;
    p.setY(y);
    d = p.distancia();
    cout << d << endl;
    return 0;
}
```

El puntero **this** es un puntero asociado a cada objeto y que apunta a si mismo.  
En este caso lo usamos para evitar ambigüedades.

# Clases en C++

- El especificador **public** define miembros públicos, que son aquéllos a los que se puede acceder por cualquier función.
- A los miembros privados que siguen al especificador **private** sólo se puede acceder por funciones miembro de la misma clase o por funciones y clases **amigas** (**friend**).
- A los miembros que siguen al especificador **protected** se puede acceder por funciones miembro de la misma clase o de clases derivadas de la misma, así como por sus **amigas**.
- Los especificadores **public**, **protected** y **private** pueden aparecer en cualquier orden. Si se omite el especificador de acceso, el acceso por defecto es **privado**.
- En la Tabla cada "x" indica que el acceso está permitido al tipo del miembro de la clase listado en la columna de la izquierda.

| Tipo de miembro | Miembro de la misma clase | Amiga | Miembro de una clase derivada | Función no miembro |
|-----------------|---------------------------|-------|-------------------------------|--------------------|
| private         | x                         | x     |                               |                    |
| protected       | x                         | x     | x                             |                    |
| public          | x                         | x     | x                             | x                  |

## Ejercicio 01

- En el ejemplo anterior implemente los siguientes métodos:
  1. El método **distancia()** que calcule la distancia entre 2 puntos. (Para ello sobrecargue el método **distancia()**)
  2. El método **perímetro()** que calcule el perímetro de un triángulo con sus 3 vértices en el plano.
  3. El método **área()** que calcule el área de un triángulo con sus 3 vértices en el plano.

Sean 3 puntos A, B y C, su área es:

$$\text{Área} = \frac{1}{2} \left| \det \begin{bmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{bmatrix} \right| = \frac{1}{2} |x_Ay_B - x_Ay_C + x_By_C - x_By_A + x_Cy_A - x_Cy_B|$$

$$\text{Área} = \frac{1}{2} |(x_A - x_C)(y_B - y_A) - (x_A - x_B)(y_C - y_A)|$$

# Herencia

- Hay tres tipos de herencia:
- La **herencia pública** significa que una clase derivada tiene acceso a los elementos públicos y protegidos de su clase base. Los elementos públicos se heredan como elementos públicos; los elementos protegidos permanecen protegidos; lo elementos privados no se heredan.
- La **herencia privada** significa que un usuario de la clase derivada no tiene acceso a ninguno de los elementos de la clase base.
- La **herencia protegida** significa que los miembros públicos y protegidos de la clase base se convierten en miembros protegidos de la clase derivada; lo elementos privados no se heredan.

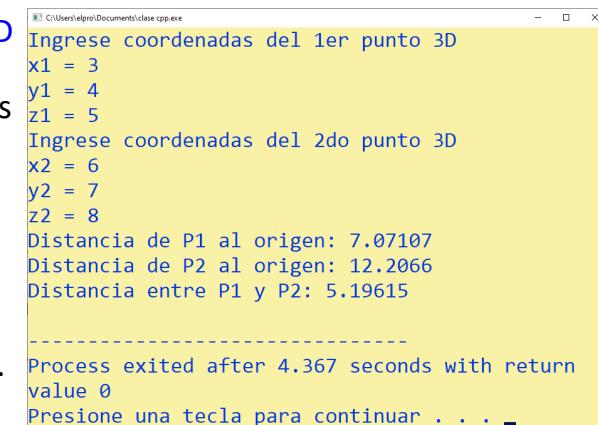
**Nota:** Por defecto la herencia es privada.

**Nota:** Normalmente, el tipo de herencia más utilizada es la herencia pública.

# Ejemplo

- Programe la clase **punto3D** que hereda de la clase **punto** las dos coordenadas **x e y**.
- El programa debe calcular la distancia del punto al origen y la distancia entre dos puntos.
- Utilice la herencia pública.  
**Sintaxis:**

```
class: Clase Derivada: public Clase Base {  
public:  
    // sección pública  
private:  
    // sección privada  
};
```



```
C:\Users\elpro\Documents\clase.cpp.exe  
Ingrese coordenadas del 1er punto 3D  
x1 = 3  
y1 = 4  
z1 = 5  
Ingrese coordenadas del 2do punto 3D  
x2 = 6  
y2 = 7  
z2 = 8  
Distancia de P1 al origen: 7.07107  
Distancia de P2 al origen: 12.2066  
Distancia entre P1 y P2: 5.19615  
-----  
Process exited after 4.367 seconds with return  
value 0  
Presione una tecla para continuar . . .
```

# Solución

```
#include <iostream>  
#include <cmath>  
using namespace std;  
class Punto  
{  
private:  
    float x, y;  
public:  
    Punto() { x = 0; y = 0; } // Constructor por defecto  
    Punto(float x, float y) //Constructor parametrizado  
    {  
        this->x = x;  
        this->y = y;  
    }  
    float getX() { return x; }  
    float getY() { return y; }  
    float distancia() { return sqrt(x * x + y * y); }  
    float distancia(Punto p)  
    {  
        return sqrt(pow(x - p.getX(), 2) + pow(y - p.getY(), 2));  
    }  
};
```

# Solución

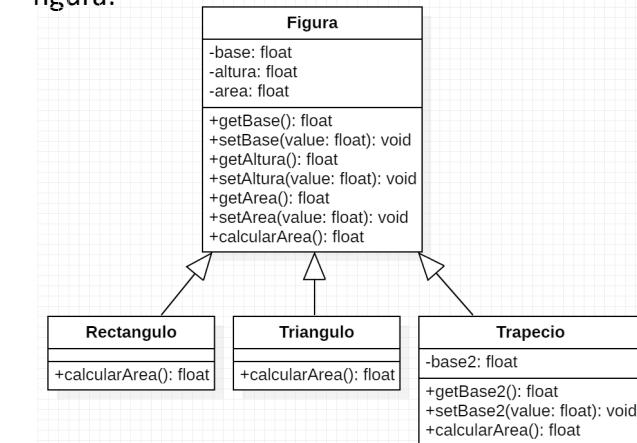
```
class Punto3D :public Punto  
{  
private:  
    float z;  
public:  
    Punto3D() :Punto() { z = 0; }  
    Punto3D(float x, float y, float z) :Punto(x, y)  
    {  
        this->z = z;  
    }  
    float distancia()  
    {  
        return sqrt(getX() * getX() + getY() * getY() + z * z);  
    }  
    float distancia(Punto3D p)  
    {  
        return sqrt(pow(getX() - p.getX(), 2) + pow(getY() - p.getY(), 2) + pow(z - p.z, 2));  
    }  
};
```

# Solución

```
int main()
{
    float x1, y1, z1, x2, y2, z2;
    cout << "Ingrese coordenadas del 1er punto 3D\n";
    cout << "x1 = ";
    cin >> x1;
    cout << "y1 = ";
    cin >> y1;
    cout << "z1 = ";
    cin >> z1;
    cout << "Ingrese coordenadas del 2do punto 3D\n";
    cout << "x2 = ";
    cin >> x2;
    cout << "y2 = ";
    cin >> y2;
    cout << "z2 = ";
    cin >> z2;
    Punto3D p1(x1, y1, z1), p2(x2, y2, z2);
    cout << "Distancia de P1 al origen: " << p1.distancia() << endl;
    cout << "Distancia de P2 al origen: " << p2.distancia() << endl;
    cout << "Distancia entre P1 y P2: " << p1.distancia(p2) << endl;
    return 0;
}
```

# Ejercicio 02

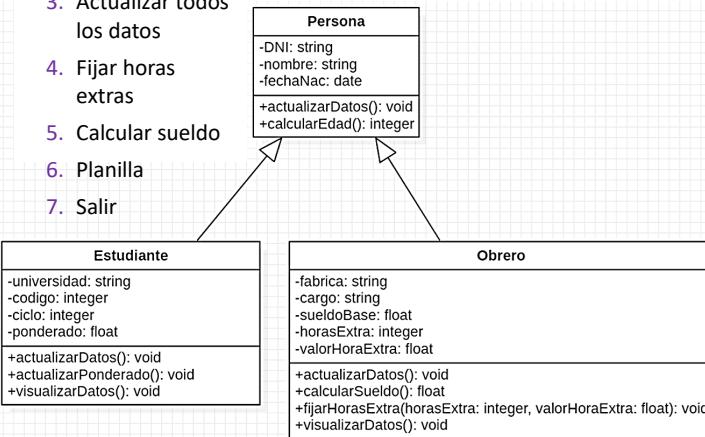
- Programe el siguiente DOO.
- El programa principal debe preguntarnos por el tipo de figura que tenemos y preguntarnos los datos necesarios para hallar el área de esa figura.



```
C:\Users\elpro\Documents\Figura.exe
Calcular areas
1. Rectangulo
2. Triangulo
3. Trapecio
4. Salir
Seleccione una opcion: 1
Ingrese base: 4
Ingrese altura: 5
El area es: 20
Calcular areas
1. Rectangulo
2. Triangulo
3. Trapecio
4. Salir
Seleccione una opcion: 2
Ingrese base: 4
Ingrese altura: 5
El area es: 10
Calcular areas
1. Rectangulo
2. Triangulo
3. Trapecio
4. Salir
Seleccione una opcion: 3
Ingrese base: 4
Ingrese altura: 5
Ingresar otra base: 10
El area es: 35
Calcular areas
1. Rectangulo
2. Triangulo
3. Trapecio
4. Salir
Seleccione una opcion: 4
-----
Process exited after 25.94 seconds with
return value 0
Presione una tecla para continuar . . .
```

# Ejercicio 03

- Programe el siguiente DOO.
- En el programa principal (**main**) cree un menú que nos permita administrar Estudiantes y Obreros:
  1. Administrar Estudiantes
  2. Administrar Obreros
  3. Salir
- La Opción 1 muestra el submenú:
  1. Ingresar estudiante
  2. Listar estudiantes
  3. Actualizar todos los datos
  4. Actualizar ponderado
  5. Salir



FIN



# FUNDAMENTOS DE PROGRAMACIÓN

## Archivo de texto

Profesor: Carlos Diaz

cdiazd@uni.edu.pe

## Archivos

- Archivos csv
- Archivos de texto
- Escribir
- Añadir
- Leer

## Archivos csv

- Los archivos CSV (del inglés comma-separated values) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por **comas** y las filas por saltos de línea. (Fuente: Wikipedia)
- Un archivo CSV (valores separados por comas) es un archivo de texto que tiene un formato específico que permite guardar los datos en un formato de tabla estructurada. (Fuente: Google Ads)

**Ejemplo:** Archivo csv abierto en bloc de notas y en Excel.

datos.csv: Bloc de notas  
Archivo Edición Formato Ver Ayuda  
Nombre, Fecha, Edad  
Juan Perez Prado, 28/04/1987, 34  
Guadalupe Escalante, 1/08/1990, 31  
Benito Smith, 6/09/1994, 27

Línea 5, columna 1 | 100% | Windows (CRLF) | UTF-8

|   | A                   | B          | C    |
|---|---------------------|------------|------|
| 1 | Nombre              | Fecha      | Edad |
| 2 | Juan Perez Prado    | 28/04/1987 | 34   |
| 3 | Guadalupe Escalante | 1/08/1990  | 31   |
| 4 | Benito Smith        | 6/09/1994  | 27   |
| 5 |                     |            |      |

## Archivos de texto

- Para crear un archivo de texto en C++ se debe incluir las librerías **iostream** y **fstream**.
- Y las clases:
  - **ofstream**: Para escribir archivos.
  - **ifstream**: Para leer archivos.
  - **fstream**: Para hacer ambas cosas, leer y escribir.

## Ejercicio 1: Escribir un archivo de texto

- Desarrolle un programa que escriba en un archivo el nombre (nombre y apellido), edad y sueldo de n personas.
- Utilice la clase `ofstream`.

**Nota:** Antes debe verificar si el archivo existe, si es el caso preguntar si desea sobreescibirlo o no.

## Ejercicio 2: Añadir a un archivo

- Escriba un programa que añada datos al archivo del ejercicio anterior.
- Utilice la clase `ofstream`.
- Para ello utilice el parámetro `ios::base::app`

**Nota:** Si el archivo no existe, este parámetro lo crea.

## Ejercicio 3: Leer a un archivo

- Escriba un programa que lea los datos del archivo del ejercicio anterior y lo muestre en pantalla con el tipo de dato correcto.
- Utilice la clase `ifstream`.

## Ejercicio 4

- Desarrolle un programa que administre una base de datos con registros de la siguiente estructura.

```
struct Persona
{
    string nombre;
    int edad;
    float sueldo;
};
```

- Debe mostrar el siguiente menú:

1. Cargar base de datos
2. Ingresar registros
4. Eliminar registros
5. Visualizar registros
6. Guardar base de datos
7. Salir

- La base de datos debe estar almacenada en el archivo `baseUNI.csv`
- Las opciones 1 y 6 son con archivos
- Las demás opciones son con estructuras.