



# Introducción a la Programación



Lenguaje C++

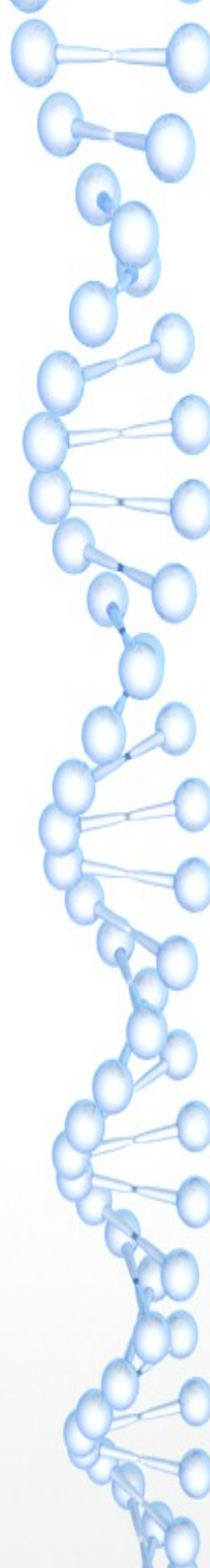


# Concepto de Algoritmo

**Según la RAE:** conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

**Los algoritmos, como indica su definición oficial, son una serie de pasos que permiten obtener la solución a un problema.**

El lenguaje algorítmico es aquel que implementa una solución teórica a un problema, indicando las **operaciones** a realizar y el **orden** en el que se deben efectuarse.

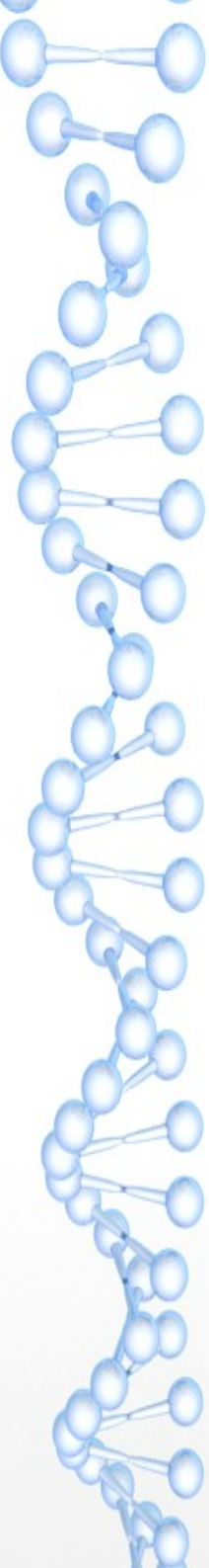


Por ejemplo en el caso de que nos encontremos en casa con una bombilla fundida en una lámpara, ¿cuál sería un posible algoritmo?

1.

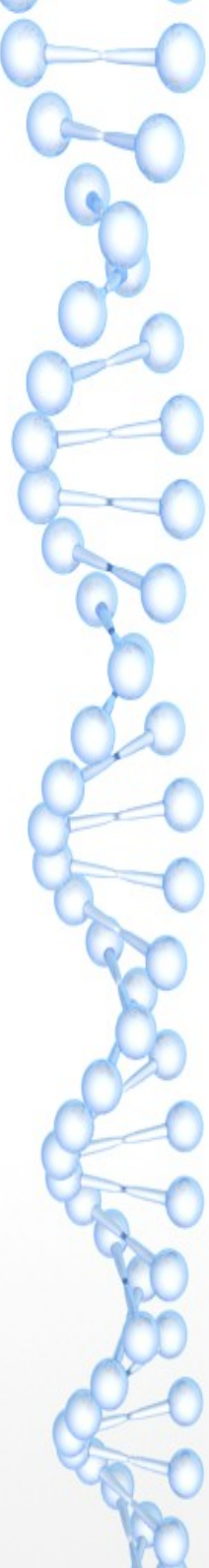
2.

3.



Por ejemplo en el caso de que nos encontremos en casa con una bombilla fundida en una lámpara, un posible algoritmo sería:

- (1) Comprobar si hay bombillas de repuesto.
- (2) En el caso de que las haya, sustituir la bombilla anterior por la nueva.
- (3) Si no hay bombillas de repuesto, bajar a comprar una nueva a la tienda y sustituir la vieja por la nueva.



Los algoritmos son la base de la programación de ordenadores, ya que los programas de ordenador se puede entender como algoritmos escritos en un código especial entendible por un ordenador.

Lo malo del diseño de algoritmos está en que no podemos escribir lo que deseemos, el lenguaje ha utilizar no debe dejar posibilidad de duda, debe recoger todas las posibilidades.



Por lo que los tres pasos anteriores pueden ser mucho más largos:

**Comprobar si hay bombillas de repuesto**

- 1. Abrir el cajón de las bombillas**
- 2. Observar si hay bombillas**

**Si hay bombillas:**

- 2.1 Coger la bombilla**
- 2.2 Coger una silla**
- 2.3 Subirse a la silla**
- 2.4 Poner la bombilla en la lámpara**

**Si no hay bombillas**

- 2.1 Abrir la puerta**
- 2.2 Bajar las escaleras....**

Cómo se observa en un algoritmo las instrucciones pueden ser más largas de lo que parecen, por lo que hay que determinar qué instrucciones se pueden utilizar y qué instrucciones no se pueden utilizar. En el caso de los algoritmos preparados para el ordenador, se pueden utilizar sólo instrucciones muy concretas.



# Características de los Algoritmos

- Un algoritmo debe resolver el problema para el que fue formulado.
- Los algoritmos son independientes del ordenador.
- Los algoritmos deben de ser precisos.
- Los algoritmos deben de ser finitos.
- Los algoritmos deben de poder repetirse.





# Elementos que Conforman un Algoritmo

**Entrada:** Los datos iniciales que posee el algoritmo antes de ejecutarse.

**Proceso:** Acciones que lleva a cabo el algoritmo.

**Salida:** Datos que obtiene finalmente el algoritmo.

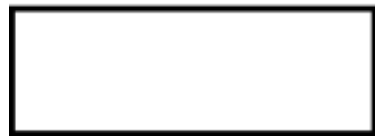


# Diagramas de Flujo

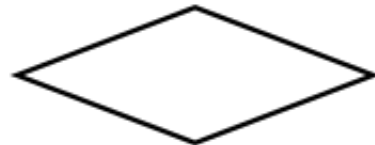
Un diagrama de flujo es la representación gráfica de un algoritmo. Estos son los elementos que podemos utilizar.



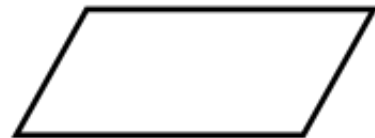
Icono de comienzo y fin del algoritmo



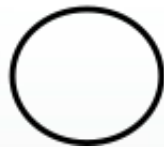
Icono de proceso (describe en su interior la acción que realiza)



Icono de decisión (dependiendo del valor de la condición el flujo del programa sigue por un lado u otro)



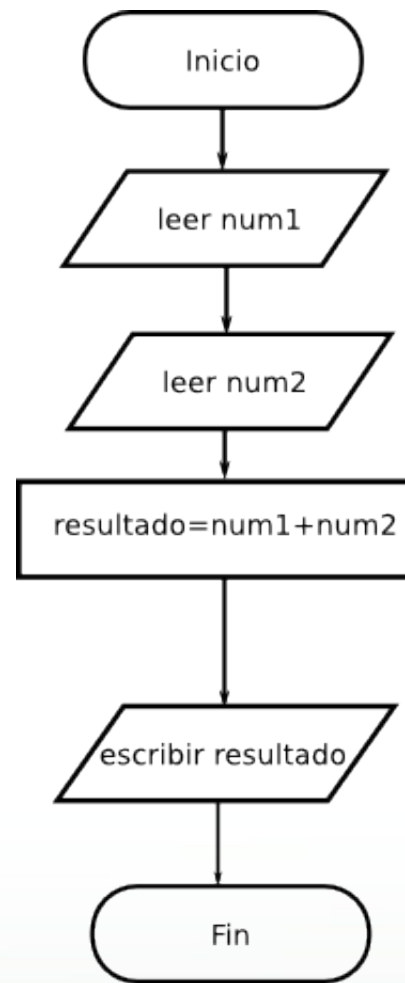
Icono de entrada/salida



Icono de conexión con otros algoritmos

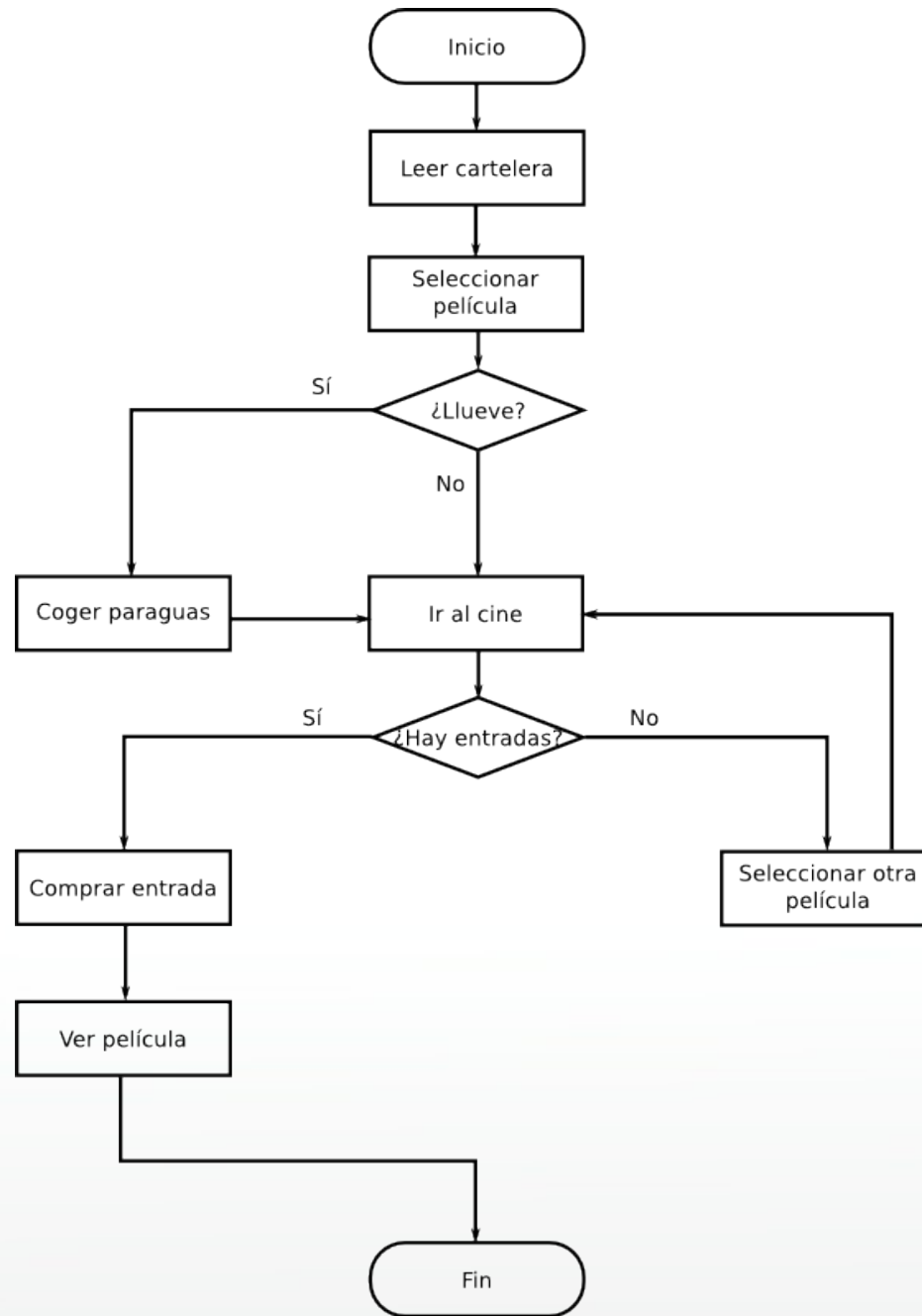
# Diagramas de Flujo

## ¿Qué hace este algoritmo?



# Diagramas de Flujo

## ¿Qué hace este algoritmo?





# ¿Qué es un Programa?

**Conjunto de instrucciones ejecutables por un ordenador**



# ¿Qué es un Lenguaje de Programación?

Las instrucciones que hemos mencionado anteriormente necesitamos escribirlas en un lenguaje de programación.

En principio el lenguaje de programación que se utilizaba era el llamado lenguaje máquina (0 y 1). Pero como podéis imaginar este tipo de lenguaje era muy complejo para las personas.

Los lenguajes de programación fueron evolucionando a la vez que las aplicaciones que se requerían eran más complejas. **Ensamblador** (2 generación de lenguajes 2GL) era una traducción del lenguaje máquina a un lenguaje algo más textual. **Lenguajes de alto nivel** (3GL) es lenguaje utilizado en la actualidad. Es más parecido al lenguaje natural, salvando las distancias, y reduce mucho las líneas de código. Pero necesita de un software adicional para traducir este lenguaje al lenguaje máquina. **4GL** son los lenguajes que se programan sin escribir casi código. **Lenguajes Orientados a Objetos** son también considerados lenguajes de 4G.

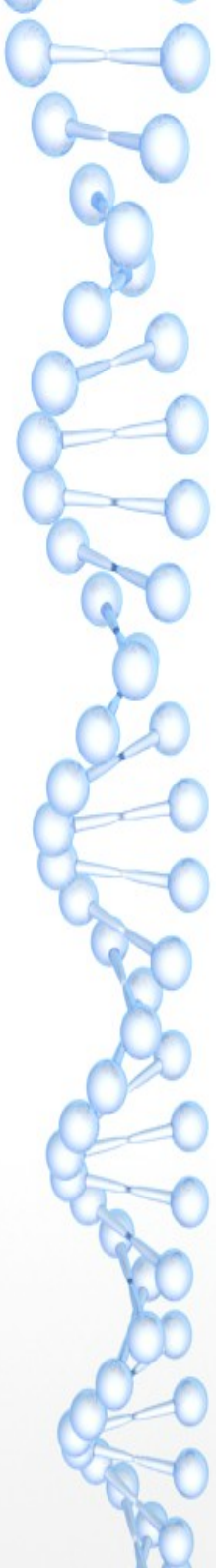


# Interpretes y Compiladores

Como hemos dicho anteriormente a partir de los lenguajes de 3G necesitamos un software especial para convertir el programa al lenguaje máquina.

Hay dos tipos: Los **Interpretes** y lo **Compiladores**.

Todo ello dependerá del lenguaje que utilizemos.



## Interprete:

En el caso de los intérpretes se convierte cada línea a código máquina y se ejecuta ese código máquina antes de convertir la siguiente línea. De esa forma si las dos primeras líneas son correctas y la tercera tiene un fallo de sintaxis, veríamos el resultado de las dos primeras líneas y al llegar a la tercera se nos notificaría el fallo y finalizaría la ejecución.

Los lenguajes integrados en páginas web son interpretados, la razón es que como la descarga de Internet es lenta, es mejor que las instrucciones se vayan traduciendo según van llegando en lugar de cargar todas en el ordenador.

Ejemplos: JavaScript, BASIC

## Compiladores:

La diferencia con los intérpretes reside en que se analizan todas las líneas antes de empezar la traducción. Una vez que ha analizado todas las líneas las traduce al lenguaje máquina creando un fichero ejecutable.

### Ventajas:

Se detectan los errores antes de ejecutar el programa.

El código máquina generado es más rápido.

Es más fácil depurar el código.

### Desventajas:

El proceso de compilación puede ser lento.

No es útil para ejecutar programas desde Internet

Ejemplos: C, C++, JAVA (semi-interpretado) y la mayoría





# Lenguaje C

## Entrada, salida y variables

Vamos a escribir nuestro primer programa y analizaremos cada instrucción:  
*hola.cpp*

```
#include <iostream>

using namespace std;

int main () {
    cout << " Hola mundo! ";
    return 0;
}
```

Salida

Hola mundo!

Sólo una línea nos interesa ahora, la que empieza por **cout**. El comando **cout** sirve para mostrar cosas por pantalla. Fijaros donde queda el cursor.



# Lenguaje C

## Entrada, salida y variables

Efectivamente, para que haya un salto de línea debemos introducir otro **cout** con un **endl (end line)**.

*hola.cpp*

```
# include < iostream >

using namespace std ;

int main () {
cout << " Hola mundo! ";
cout<<endl;
return 0;
}
```

Efectivamente, para que haya un salto de línea debemos introducir otro **cout** con un **endl (end line)**.



# Lenguaje C

## Entrada, salida y variables

Otro ejemplo:

*calculos1.cpp*

```
# include < iostream >

using namespace std ;

int main () {
cout << " Unos calculos : " << endl ;
cout << " 2+8*(3+1)= " ;
cout << " 2+8* " << 3+1 << " = " ;
cout << " 2+ " << 8*(3+1) << " = " ;
cout << 2+8*(3+1) << endl;
return 0;
}
```

Salida

Unos calculos:  
 $2+8*(3+1) = 2+8*4 = 2+32 = 34$



# Lenguaje C

## Entrada, salida y variables

Vamos a analizar el programa anterior:

```
# include < iostream >

using namespace std ;

int main () {
cout << " Unos calculos : " << endl ;
cout << " 2+8*(3+1)= " ;
cout << " 2+8* " << 3+1 << " = " ;
cout << " 2+ " << 8*(3+1) << " = " ;
cout << 2*8*(3+1) << endl;
return 0;
}
```

- Todas las instrucciones acaban en punto y como ;
- Un conjunto de instrucciones forman un **bloque** y van entre llaves { }.
- Podemos escribir varias cosas en el mismo **cout** siempre que se separe convenientemente por los símbolos <<
- Podemos escribir cadenas de texto (string) siempre que las pongamos entre comillas "" y aparecerán en pantalla tal cual.
- Podemos escribir expresiones matemáticas (evidentemente sin poner comillas) y lo que aparecerá en pantalla será el resultado de dicha expresión (por ejemplo: `cout << 8*(3+1);` *aparece en pantalla 32*).
- **endl** como hemos dicho anteriormente significa salto de línea.



# Lenguaje C

## Entrada, salida y variables

*calculos2.cpp*

```
# include <iostream>

using namespace std ;

int main () {
    int x=5;
    int y=3;

    cout<<"El numero x es "<< x <<" y su cuadrado es "<< x*x <<endl;
    cout<<"El numero y es "<< y <<" y su cuadrado es "<< y*y <<endl;
    return 0;
}
```

### Salida

El numero x es 5 y su cuadrado es 25

- **x** e **y** son variables. Es decir lo que hace esta instrucción (*int x; int y;*) es decirle al ordenador que reserve un trozo de memoria para guardar, en este caso, un número entero. A partir de ahora esos trozos de memoria se llamarán **x** e **y** respectivamente.
- Cambia el valor de **x** y el valor de **y** ¿qué ha pasado?.



# Lenguaje C

## Entrada, salida y variables

### Tipos de variables

Tipo de Datos	Descripción	Número de bytes
int	número entero	4
float	número real	4
double	real doble	8
char	carácter	1
string	cadena de caracteres (texto)	según la longitud del texto

- antes de utilizar una variable debemos **definirla**, para ello haremos la siguiente instrucción:

*tipo\_de\_datos nombre;*

Ejemplos:

```
char a; //variable del tipo char sin ningún valor inicial
```

```
int x; //variable del tipo int sin ningún valor inicial
```

```
char a='a'; //variable del tipo char con el valor inicial a
```

```
int y=23; //variable del tipo int con el valor inicial 23
```

```
float f=3.141516; //variable del tipo float con un valor inicial de 3.141516
```

# Lenguaje C

## Entrada, salida y variables

*datos\_personales.cpp*

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

// esto es un comentario de una línea

int main () {
    cout<<"Hola, ¿como te llamas? ";
    string nombre;
    cin>>nombre;
    cout<<"Hola, "<<nombre<<" ¿en que año naciste?"<<endl;
    int nac;
    cin>>nac;
    cout<<nombre<<" con ese careto que tienes debes tener "
    <<2016-nac-1<<" o "<<2016-nac
    <<" según en que mes hayas nacido"<<endl;
    return 0;
}
```





# Lenguaje C

## Entrada, salida y variables

### Salida

```
Hola, ¿como te llamas? Hommer  
Hola, Hommer ¿en que año naciste?  
1998  
Hommer con ese careto que tienes debes tener 17 o 18 años, según en que mes hayas nacido
```

Vamos a analizar el programa:

- En este programa hay dos variables **nombre** y **nac**.
- La variable **nombre** es del tipo string y guarda una cadena de texto.
- Fijaos que al principio hemos añadido una nueva línea, **#include <string>**, siempre que queramos utilizar una variable del tipo string deberemos ponerla.
- **#include <iostream>** es para indicar que vamos a utilizar **cout** o/y **cin**.
- En el programa las dos variables se definen sin inicializar, es decir sin asignarles ningún valor, porque se leerán a continuación con **cin**.
- **cin** funciona igual que **cout** excepto que utilizaremos **>>** ya que **cin** pone datos en una variable, en vez de sacarlos **<<**.

Por tanto **cout <<** espera recibir valores para mostrarlos mientras **cin >>** espera recibir valores para guardarlos en variables



# Lenguaje C

## Entrada, salida y variables

### Operadores Aritméticos

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto
=	Asignación

Estos son los operadores que podemos utilizar para operaciones con números constantes (ej. `cout<<7+5;`) o con variables ( `resul=a+b;` ).

Veamos un ejemplo.

# Lenguaje C

## Entrada, salida y variables

*variables.cpp*

```
# include <iostream>

using namespace std ;

// cambiar el valor almacenado en una variable

int main () {

    cout<<"Dime valor para almacenar en la variable 'número' "<<endl;
    int numero;
    cin>>numero;
    cout<<"el valor almacenado en 'número' es "<<numero<<endl;
    numero=numero*2;
    cout<<"he hecho un cambio en el valor almacenado y 'numero' es "<<numero<<endl;
    numero=numero+5;
    cout<<"ahora le he sumado al valor almacenado 5 y 'numero' es "<<numero<<endl;
    numero=0;
    cout<<"y ahora le he asignado el valor 0 a la variable"<<endl;
    cout<<"la variable 'numero' es "<<numero<<endl;
    return 0;

}
```



# Lenguaje C

## Entrada, salida y variables

Dime valor para almacenar en la variable 'número'

75

el valor almacenado en 'número' es 75

he hecho un cambio en el valor almacenado y 'numero' es 150

ahora le he sumado al valor almacenado 5 y 'numero' es 155

y ahora le he asignado el valor 0 a la variable

la variable 'numero' es 0

Analizamos el programa:

- Definimos la variable **numero** para almacenar con **cin** el número que nos introduzca el usuario.
- Con la instrucción `numero=numero*2;` lo que hacemos es multiplicar por 2 el valor que tenga la variable `numero` y ese resultado lo almacenamos en la propia variable `numero`. (Ahora número vale el doble que al principio)
- Con la instrucción `numero=numero+5;` al valor de la variable `numero` le sumamos 5.
- Con la instrucción `numero=0;` estamos asignándole el valor 0 a la variable `numero`.
- Todo ello lo vamos mostrando con la instrucción **cout**.
- Observar que no podemos utilizar una variable sin que previamente la hayamos definido.



# Lenguaje C

## Toma de Decisiones

Hasta ahora todos los programas son secuenciales, todas las instrucciones se ejecutan en el orden establecido. Pero como podemos imaginar los algoritmos deberán ejecutar unas instrucciones u otras según unas condiciones establecidas o deberán repetir la ejecución de esas mismas instrucciones un número  $n$  de veces.

Para esto existen las ***instrucciones de control*** que son las encargadas de controlar la secuencia del programa.

Para ello utilizaremos los ***Operadores relacionales***:

Operador	Operación
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que
==	igual
!=	distinto de (no igual)



# Lenguaje C

## Toma de Decisiones. If...else

*Si\_es\_Cero.cpp*

```
#include <iostream>

using namespace std;

/* esto es un comentario de varias líneas
   esto es un comentario de varias líneas
   esto es un comentario de varias líneas */
int main()
{
    int n;

    cout<<"Escribe un número: ";
    cin>>n;

    if(n==0)
        cout<<"Has escrito el número 0."<<endl;

    else
        cout<<"Has escrito un número distinto de 0."<<endl;

    return 0;
}
```



# Lenguaje C

## Toma de Decisiones. If...else

Salida 1

Escribe un número: 2  
Has escrito un número distinto de 0.

Salida 2

Escribe un número: 0  
Has escrito el número 0.

Analizamos el programa:

- Como podemos ver este es el primer programa hasta ahora que tiene dos salidas posibles. Esto es gracias a la instrucción ***if...else*** que se llama instrucción condicional.
- Se emplea de la siguiente manera: ***if (condición) instrucción;  
else instrucción;***
- Como dijimos anteriormente un conjunto de instrucciones forman un bloque y deberá ir entre { }. Por lo tanto:





# Lenguaje C

## Toma de Decisiones. If...else

```
if (condición){  
    instrucción_1;  
    instrucción_2;  
    instrucción_3;  
    ...  
    instrucción_n;  
}  
Else{  
    instrucción_1;  
    instrucción_2;  
    instrucción_3;  
    ...  
    instrucción_n;  
}
```

- Como podéis imaginar una parte fundamental para que el algoritmo funcione correctamente es saber construir la condición. Para ello utilizaremos los operadores condicionales que hemos descrito en la tabla anterior. En este caso la **condición** es  **$n==0$** , las condiciones siempre son preguntas, en este caso, ¿n es igual que 0? si la respuesta es afirmativa solo se harán la o las instrucciones del if, si la respuesta es negativa solo se harán la o las instrucciones del else.



# Lenguaje C

## Toma de Decisiones. If...else Encadenar condiciones

Las condiciones se pueden encadenar con “y” o con “o”, de esta forma podemos formar expresiones que se evaluarán a verdadero o falso.

Operador	Significado
&&	Y
	O
!	No



# Lenguaje C

## Toma de Decisiones. If...else Encadenar condiciones

*nota.cpp*

```
#include <iostream>

using namespace std;

/* programa que dada una nota de entrada, que guardaremos en la variable n,
nos dice:
    Si la nota es menor que 5 y mayor que cero nos dice que es SUSPENSO.
    Si la nota es mayor o igual que 5 y menor o igual que 10 APROBADO
    En otro caso nos dice que la nota es INCORRECTA*/

int main()
{
    int n;

    cout<<"Introduzca la nota: ";
    cin>>n;

    if(n>=0 && n<5)
        cout<<"SUSPENSO"<<endl;

    else if(n>=5 && n<=10)
        cout<<"APROBADO"<<endl;
    else
        cout<<"la nota es INCORRECTA"<<endl;

    return 0;
}
```



# Lenguaje C

Toma de Decisiones. If...else Encadenar condiciones

Salida 1

Introduzca la nota: 3  
SUSPENSO

Salida 2

Introduzca la nota: 8  
APROBADO

Salida 3

Introduzca la nota: 11  
la nota es INCORRECTA



# Lenguaje C

## Toma de Decisiones. If...else Encadenar condiciones

Vamos a analizar el programa:

- En este programa vemos que las condiciones son dobles, ya que, para ser suspenso la nota debe estar entre 0 y 4. Para ello la **condición\_1** la hemos unido con && (y) con la **condición\_2** de tal forma que si se cumplen las dos la expresión es verdadera, si una de las dos no se cumple la expresión es falsa. Igual para el caso de aprobado
- Como veis este programa tiene tres posibles salidas, por ello, hemos introducido una nueva instrucción (**else if**). Podemos llamarlo if compuesto:

```
    If (condición){  
        ...  
    }  
    else if (condición){  
        ...  
    }  
    else if (condición){  
        ...  
    }  
    ....  
(tantos else if como necesite)  
    else{  
        ...  
    }
```

# Lenguaje C

## Toma de Decisiones. switch

Cuando queremos analizar los posibles valores de una misma variable podemos utilizar esta instrucción. Es muy típica para crear los menús de opción.

*nota2.cpp*

```
#include <iostream>
using namespace std;
/* programa que dada una nota (entera) de entrada, que guardaremos en la variable n,
nos dice: SUSPENSO, SUFICIENTE, BIEN, NOTABLE O SOBRESALIENTE*/
int main()
{
    int nota;

    cout<<"Introduzca la nota (Debes poner un número entero): ";
    cin>>nota;

    if(nota>=0 && nota<5)
        cout<<"SUSPENSO"<<endl;
    else{
        switch(nota){
            case 5:
                cout<<"SUFICIENTE"<<endl;
                break;
            case 6:
                cout<<"BIEN"<<endl;
                break;
            case 7:
                cout<<"NOTABLE"<<endl;
                break;
            case 8:
                cout<<"NOTABLE"<<endl;
                break;
            case 9:
                cout<<"SOBRESALIENTE"<<endl;
                break;
            case 10:
                cout<<"MATRICULA DE HONOR"<<endl;
                break;
            default:
                cout<<"NOTA INCORRECTA"<<endl;
        }
    }
    return 0;
}
```



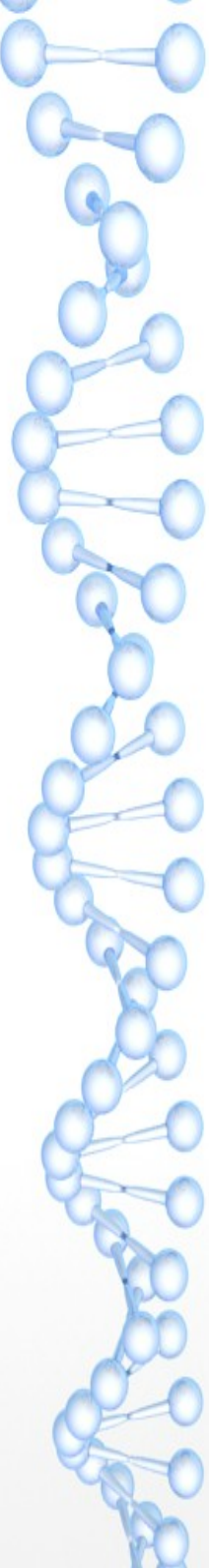
# Lenguaje C

## Toma de Decisiones. switch

Vamos a analizar el programa:

- El programa pide una nota, fíjate que como la variable la hemos definido del tipo **int** los decimales se despreciarán.
- Hacemos una instrucción **if...else** porque todos los valores del suspenso dan la misma salida (SUSPENSO) y en el caso del **else** hacemos el **switch** para analizar todos los posibles valores que nos quedan.
- El **switch** debe tener un caso limitado de valores.
- Siempre evaluamos una variable que irá entre paréntesis. En nuestro caso la variable nota: **Switch (variable)**
- Los valores posibles se indican en cada **case** y acaba en dos puntos (:). En nuestro programa los valores 5, 6, 7, 8, 9 y 10: **case 1:**
- No se puede analizar dos valores a la vez. **Case 6 && 8: (incorrecto)**
- Las instrucciones a ejecutar en cada caso deben acabar la instrucción **break;**
- Se puede poner como último caso **default:** que podemos interpretar como “por defecto”. Es decir en nuestro caso sería para los valores mayores de 10 y menores de 0 o dicho de otra manera para todas las notas incorrectas.





# Lenguaje C

## Bucles. While

La programación es secuencial, pero eso no significa que las condiciones no se puedan comprobar más de una vez.

*nota3.cpp*

```
#include <iostream>

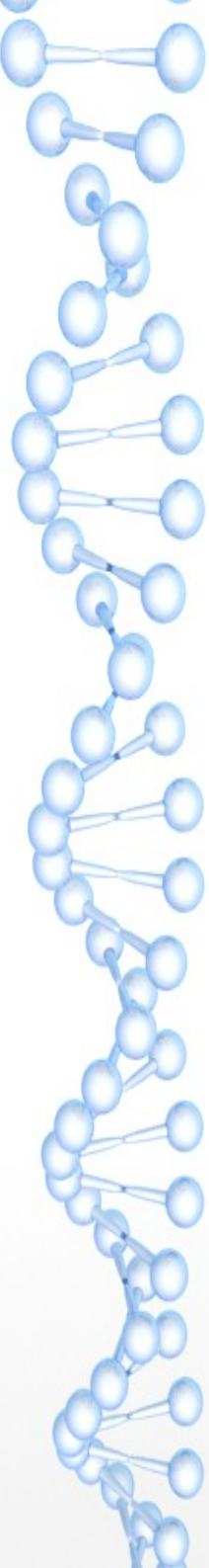
using namespace std;

/* Programa para introducir la nota de Informática si la nota no es
correcta nos lo indica y no la vuelve a pedir otra vez*/

int main()
{
    int nota;

    cout<<"Introduzca la nota de Informática (Debes poner un númro entero): ";
    cin>>nota;

    while(nota<0 || nota>10) // Mientras que la nota sea < que 0 o > que 10 ejecuta las instrucciones del While
    {
        cout<<nota<<" no es correcta!!!"<<endl;
        cout<<"Introduzca una nota correcta: ";
        cin>>nota;
    }
    cout<<"Has sacado un "<<nota<<" en Informatica..."<<endl;
    return 0;
}
```



# Lenguaje C

## Bucles. While

Salida

```
Introduzca la nota de Informática (Debes poner un número entero): 79
79 no es correcta!!!
Introduzca una nota correcta: -8
-8 no es correcta!!!
Introduzca una nota correcta: 10
Has sacado un 10 en Informática...
```

Vamos a analizar el programa:

- Este programa nos pide una nota para la asignatura de informática, con la particularidad que si la nota no es correcta nos lo comunica y nos la vuelve a pedir otra vez, así hasta que le introduzcamos una nota correcta.
- Esto se consigue utilizando el comando ***while (...){...}***. El código dentro de las llaves se ejecutará una y otra vez *mientras* la condición de los paréntesis sea cierta.
- En nuestro ejemplo se ejecutará el código dentro de las llaves, es decir, el mensaje de “nota incorrecta” y el volver a pedir otra nota mientras que la condición de  $\text{nota} < 0$  o  $\text{nota} > 10$  sea verdadera, mientras que la nota sea negativa o mayor que 10.
- En C++, una condición falsa da como resultado el valor 0 y condición verdadera da un valor distinto de 0.
- Fijaros que si el primer valor que introducimos en el programa es una nota correcta, la condición del while será falsa y por ello ni siquiera se entrará al bloque del while y el programa finalizará mostrando la nota de informática.



# Lenguaje C

## Bucles. do-While

El while tiene dos formatos, uno en el que se desea comprobar la condición al principio y otra en que la comprobación es al final (**do-while**).

nota4.cpp

```
#include <iostream>

using namespace std;

/* Programa para introducir la nota de Informática si la nota no es
correcta nos lo indica y nos la vuelve a pedir otra vez*/

int main()
{
    int nota;

    do
    {
        cout<<"Introduzca la nota de Informática (Debes poner un númro entero): ";
        cin>>nota;

        if(nota<0 || nota>10) //comprobamos si la nota es incorrecta
            cout<<nota<<" no es correcta!!!"<<endl;
    }while(nota<0 || nota>10);

    cout<<"Has sacado un "<<nota<<" en Informatica..."<<endl;
    return 0;
}
```



# Lenguaje C

## Bucles. do-While

Salida

Introduzca la nota de Informática (Debes poner un número entero): -9  
-9 no es correcta!!!

Introduzca la nota de Informática (Debes poner un número entero): 89  
89 no es correcta!!!

Introduzca la nota de Informática (Debes poner un número entero): 9  
Has sacado un 9 en Informática...

Vamos a analizar el programa:

- Como vemos el resultado es igual que el programa anterior.
- La diferencia está en que el comando **do-while** al comprobar la condición al final, estamos seguros que las instrucciones del comando **do** se van ejecutar al menos una vez (1 a n veces se puede ejecutar el **do** en general). Si la primera nota es 5, *por ejemplo*, la condición del **while** es falsa y no se vuelve a ejecutar el **do**, se acaba el programa, si la primera nota es -4 la condición del **while** es verdadera y se vuelve a ejecutar el **do** así n veces hasta que la condición del **while** sea falsa.
- En el programa *nota3.cpp* el **while** puede que no se ejecuta nunca ya que si la primera nota es por ejemplo, un 7 la condición del **while** es falsa y no se ejecutan las instrucciones entre llaves, como hemos dicho en la explicación anterior. El **while** se ejecuta de 0 a n veces.



# Lenguaje C

## Bucles. Contadores. **for**

Cuando se desea crear un contador, se suele utilizar el comando **for**.

contador.cpp

```
#include <iostream>

using namespace std;
//Programa que escribir numeros en pantalla del 0 al 10
int main()
{
    int x;

    for(x=0;x<10;x++)
        cout <<" "<<x; //escribe un espacio y luego el número
    cout << endl; // cuando sale del for hacemos un salto de línea
    return 0;
}
```



# Lenguaje C

## Bucles. Contadores. **for**

Salida

0 1 2 3 4 5 6 7 8 9

Vamos a analizar el programa:

- Los contadores sirven para ejecutar un líneas de código un número de veces concreto.
- Los contadores se pueden ser de: **Incremento o Decremento**, dependerá de la instrucción **x++** o **x--** (puede ser x o la variable (int) que nosotros hayamos definido).
- La sintaxis del for es:

```
for(inicialización;condición;incremento)  
{  
    Instrucción1;  
    ...  
    InstrucciónN;  
}
```

***inicialización:*** aquí le asignaremos un número inicial a la variable contador, por ejemplo x=0.

***condición:*** es la condición de parada, es decir, mientras que la condición sea verdadera se ejecutarán las instrucciones del for, cuando sea falsa parará.

***incremento o decremento:*** la variable contador se tiene que incrementar o decrementar para contar las veces que se ejecutan las instrucciones del **for**.





# Lenguaje C

## Bucles. Contadores. **for**

### ***Ejercicios***

- Modifica el programa anterior para que escriba número del 0 hasta 100. Hasta el 100. Comenta en el código del programa qué es lo que hemos tenido que cambiar. Guardalo como contador1.cpp
- Modifica el programa anterior para que sea el usuario el que le diga al programa hasta donde tiene que contar. Explica (con un comentario dentro del fichero cpp) que es lo que has tenido que modificar. La posible salida sería. Guardalo como contador2.cpp

*salida*

```
¿Hasta dónde quieres contar? 20
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

- Modifica el programa contador1.cpp para que haga una cuenta atrás desde 100 hasta 0. Explica (con un comentario dentro del fichero cpp) que es lo que has tenido que modificar. Guardalo como contador3.cpp



# Lenguaje C

## Estructura básica de datos. **arrays**

En los programas de notas hemos trabajado siempre con una nota. Ahora vamos a ver como almacenar muchos datos en una misma variable.

*nota\_media.cpp*

```
#include <iostream>

using namespace std;
//Programa que pide 5 notas y calcula la nota media utilizando arrays
int main()
{
    int nota[5]; //declaración del arrays de 5 int (5 enteros)
    int media=0; //variable de tipo entero para calcular la nota media
    int x; // variable contador

    //pedimos las cinco notas de los exámenes y las almacenamos en cada posición del array (de la 0 a la 4)
    for(x=0;x<5;x++)
    {
        cout <<"Escribe la nota del examen "<<x+1<<" ";
        cin >> nota[x];
    }

    //para obtener la media sumamos todas las notas que tenemos almacenadas en el array
    for (x = 0; x < 5; x++)
        media=media+nota[x];

    //mostramos en pantalla la suma de la media dividido entre 5, es decir, la media de las notas
    cout << "la media que has sacado es: "<<media/5<<endl;

    return 0;
}
```

# Lenguaje C

## Estructura básica de datos. **arrays**

Salida

```
Escribe la nota del examen 1 4
Escribe la nota del examen 2 10
Escribe la nota del examen 3 3
Escribe la nota del examen 4 6
Escribe la nota del examen 5 7
la media que has sacado es: 6
```

- Los arrays son un conjunto de elementos del mismo tipo. Estos elementos tendrán todos el mismo nombre y ocuparán un espacio contiguo en la memoria

representación del  
array. ***int nota [5];***

0	1	2	3	4
4	10	3	6	7

posiciones

contenido

- El primer elemento se almacena en la posición 0 y el último en la posición n-1, siendo n el tamaño del array.
- Cundo definimos un array debemos poner ***tipo\_datos nombre[tamaño];*** en nuestro ejemplo el array es de enteros, se llama nota y es de tamaño 5. Donde el primer elemento es ***nota[0]*** y el último ***nota[4]***



# Lenguaje C

## Estructura básica de datos. **arrays**

- Para escribir en un array o para leer los datos que hay almacenados en él se hace con la ayuda del **for**.
- No hace falta que los arrays tenga valores iniciales. Pero si se conocemos los valores iniciales se pueden especificar entre llaves:

```
Int notas[5]={4, 10, 3, 6, 7};
```

- En nuestro ejemplo hemos hecho dos **for** uno para asignar las notas de los exámenes y otro para sumar esas notas en la variable media. Fijaros que para *recorrer* un array utilizamos un **for** que va desde la posición  $x=0$  hasta la posición  $x=4$  (por eso la condición es  $x<5$  no  $x\leq 5$ ).
- Para hacer la suma de las notas tenemos que inicializar la variable media a 0, y se hace con la instrucción dentro del **for**

```
for(x=0;x<5;x++)  
    media=media+nota[x];
```

**Cuando x es 0 nota[0] es 4 entonces media = 0+ 4= 4**

**Cuando x es 1 nota[1] es 10 entonces media= 4+10= 14**

**Cuando x es 2 nota[2] es 3 entonces media= 14+3= 17**

**Cuando x es 3 nota[3] es 6 entonces media= 17+6= 23**

**Cuando x es 4 nota[4] es 7 entonces media= 23+7= 30**

# Lenguaje C

## Estructura básica de datos. Matrices con **Arrays**

- Todos sabéis lo que son las matrices en matemáticas. Es un tabla de números dispuestos en filas y columnas:

1	5	6
7	9	10
4	10	3

3x3

- Esto es una matriz de 3x3 3 filas y 3 columnas
- Esto se puede representar con arrays de la siguiente manera:

```
Int matriz [3][3]; //esto es una matriz de 3 filas y 3 columnas sin inicializar
Int matriz [3][3]={
```

```
    {1, 5, 6},
    {7, 9, 10},
    {4, 10, 3}
};
```



# Introducción a la Programación

## Lenguaje C++



**Fin**