



Facultad de Informática  
Universidad Complutense

Apuntes de clase de la asignatura

# Fundamentos de la programación

1º curso

Grado en Ingeniería en Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

2013-2014

Luis Hernández Yáñez



Licencia *Creative Commons*:  
Reconocimiento, No comercial y Compartir igual.  
<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Esta publicación contiene los apuntes de clase de la asignatura Fundamentos de la programación, asignatura de 1º curso de los grados que se imparten en la Facultad de Informática de la UCM.

Durante los tres primeros cursos en los que se ha impartido la asignatura, este material ha sido sometido a continuas revisiones y contribuciones por parte de los profesores que han impartido los distintos grupos de la asignatura. Aunque el trabajo ha quedado bastante consolidado, estoy seguro de que todavía contiene muchas erratas. Si encuentras alguna, no dudes, por favor, en hacérmelo saber y conseguir así que la siguiente versión esté mejor depurada.

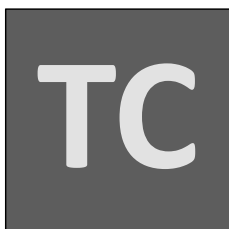
Quiero agradecer a todos los profesores que han impartido la asignatura su contribución en el desarrollo del material, destacando especialmente la labor de Pablo Moreno Ger y Carlos Cervigón Rückauer.

Luis Hernández Yáñez  
Profesor de la Facultad de Informática de la UCM



# Fundamentos de la programación

---



## Índice general

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice de temas

---

Tema 1	Computadoras y programación	1
Tema 2	Tipos e instrucciones I	48
	Anexo: Detalles técnicos de los tipos	212
Tema 3	Tipos e instrucciones II	225
	Anexo I: El operador ternario ?	398
	Anexo II: Ejemplos de secuencias	402
Tema 4	La abstracción procedimental	425
	Anexo: Más sobre subprogramas	496
Tema 5	Tipos de datos estructurados	512
	Anexo: Cadenas al estilo de C	580
Tema 6	Recorrido y búsqueda en arrays	588
Tema 7	Algoritmos de ordenación	649
	Anexo: Más sobre ordenación	742
Tema 8	Programación modular	755
	Anexo: Ejemplo de modularización	832
Tema 9	Punteros y memoria dinámica	847
	Anexo: Punteros y memoria dinámica	938
Tema 10	Introducción a la recursión	981
	Apéndice: Archivos binarios	1049



# Tema 1: Computadoras y programación

---

Informática, computadoras y programación	3
Lenguaje máquina y ensamblador	12
Lenguajes de programación de alto nivel	15
Un poco de historia	19
Programación e Ingeniería del Software	24
El lenguaje de programación C++	27
Sintaxis de los lenguajes de programación	30
Un primer programa en C++	35
Herramientas de desarrollo	39
C++: Un mejor C	45

Luis Hernández Yáñez



Fundamentos de la programación



## Tema 2: Tipos e instrucciones I

---

Un ejemplo de programación	50	Operadores relacionales	177
El primer programa en C++	64	Toma de decisiones (if)	180
Las líneas de código del programa	80	Bloques de código	183
Cálculos en los programas	86	Bucles (while)	186
Variables	92	Entrada/salida por consola	190
Expresiones	98	Funciones definidas por el programador	199
Lectura de datos desde el teclado	108		
Resolución de problemas	119		
Los datos de los programas	127		
Identificadores	129		
Tipos de datos	133		
Declaración y uso de variables	142		
Instrucciones de asignación	147		
Operadores	152		
Más sobre expresiones	160		
Constantes	167		
La biblioteca <code>cmath</code>	171		
Operaciones con caracteres	174		

Luis Hernández Yáñez



Fundamentos de la programación



## Tema 2 (Anexo): Detalles técnicos de los tipos

---

int	214
float	216
Notación científica	217
double	218
char	220
bool	221
string	222
Literales con especificación de tipo	223



## Tema 3: Tipos e instrucciones II

---

Tipos, valores y variables	227	El bucle for	321
Conversión de tipos	232	Bucles anidados	331
Tipos declarados por el usuario	236	Ámbito y visibilidad	339
Tipos enumerados	238	Secuencias	349
Entrada/Salida		Recorrido de secuencias	355
con archivos de texto	248	Secuencias calculadas	363
Lectura de archivos de texto	253	Búsqueda en secuencias	370
Escritura en archivos de texto	266	Arrays de datos simples	374
Flujo de ejecución	272	Uso de variables arrays	379
Selección simple	276	Recorrido de arrays	382
Operadores lógicos	282	Búsqueda en arrays	387
Anidamiento de if	286	Arrays no completos	393
Condiciones	290		
Selección múltiple	293		
La escala if-else-if	295		
La instrucción switch	302		
Repetición	313		
El bucle while	316		



## Tema 3: Anexos

---

### Anexo I: El operador ternario ?

El operador ternario ?	399
------------------------	-----

### Anexo II: Ejemplos de secuencias

Recorridos	404
Un aparcamiento	405
¿Paréntesis bien emparejados?	409
¿Dos secuencias iguales?	412
Números primos menores que N	413
Búsquedas	417
Búsqueda de un número en un archivo	419
Búsquedas en secuencias ordenadas	420



## Tema 4: La abstracción procedimental

---

Diseño descendente: Tareas y subtareas	427
Subprogramas	434
Subprogramas y datos	441
Parámetros	446
Argumentos	451
Resultado de la función	467
Prototipos	473
Ejemplos completos	475
Funciones de operador	477
Diseño descendente (un ejemplo)	480
Precondiciones y postcondiciones	490



## Tema 4 (Anexo): Más sobre subprogramas

---

Archivos como parámetros	498
La función <code>main()</code>	501
Argumentos implícitos	504
Sobrecarga de subprogramas	508



## Tema 5: Tipos de datos estructurados

---

Tipos de datos	514
Arrays de nuevo	517
Arrays y bucles <code>for</code>	520
Más sobre arrays	522
Inicialización de arrays	523
Enumerados como índices	524
Paso de arrays a subprogramas	525
Implementación de listas	528
Cadenas de caracteres	531
Cadenas de caracteres de tipo <code>string</code>	535
Entrada/salida con <code>string</code>	539
Operaciones con <code>string</code>	541
Estructuras	543
Estructuras dentro de estructuras	549
Arrays de estructuras	550
Arrays dentro de estructuras	551
Listas de longitud variable	552
Un ejemplo completo	558
El bucle <code>do..while</code>	562



## Tema 5 (Anexo): Cadenas al estilo de C

---

Cadenas al estilo de C	582
E/S con cadenas al estilo de C	583
La biblioteca <code>cstring</code>	584
Ejemplo	585



## Tema 6: Recorrido y búsqueda en arrays

---

Recorrido de arrays	590
Arrays completos	593
Arrays no completos con centinela	594
Arrays no completos con contador	595
Ejemplos	597
Generación de números aleatorios	601
Búsquedas en arrays	604
Arrays completos	606
Arrays no completos con centinela	607
Arrays no completos con contador	608
Ejemplo	610
Recorridos y búsquedas en cadenas	614
Más ejemplos de manejo de arrays	617
Arrays multidimensionales	630
Inicialización de arrays multidimensionales	638
Recorrido de un array bidimensional	641
Recorrido de un array N-dimensional	644
Búsqueda en un array multidimensional	647





## Tema 7: Algoritmos de ordenación

---

Algoritmos de ordenación	651
Algoritmo de ordenación por inserción	654
Ordenación de arrays por inserción	665
Algoritmo de ordenación por inserción con intercambios	672
Claves de ordenación	680
Estabilidad de la ordenación	688
Complejidad y eficiencia	692
Ordenaciones naturales	694
Ordenación por selección directa	701
Método de la burbuja	716
Listas ordenadas	722
Búsquedas en listas ordenadas	729
Búsqueda binaria	731



## Tema 7 (Anexo): Más sobre ordenación

---

Ordenación por intercambio	744
Mezcla de dos listas ordenadas	747



# Tema 8: Programación modular

---

Programas multiarchivo y compilación separada	757
Interfaz frente a implementación	762
Uso de módulos de biblioteca	768
Ejemplo: Gestión de una lista ordenada I	770
Compilación de programas multiarchivo	778
El preprocesador	780
Cada cosa en su módulo	782
Ejemplo: Gestión de una lista ordenada II	784
El problema de las inclusiones múltiples	789
Compilación condicional	794
Protección frente a inclusiones múltiples	795
Ejemplo: Gestión de una lista ordenada III	796
Implementaciones alternativas	804
Espacios de nombres	808
Implementaciones alternativas	817
Calidad y reutilización del software	827



# Tema 8 (Anexo): Ejemplo de modularización

---

Modularización de un programa	833
-------------------------------	-----



## Tema 9: Punteros y memoria dinámica

---

Direcciones de memoria y punteros	849
Operadores de punteros	854
Punteros y direcciones válidas	864
Punteros no inicializados	866
Un valor seguro: NULL	867
Copia y comparación de punteros	868
Tipos puntero	873
Punteros a estructuras	875
Punteros a constantes y punteros constantes	877
Punteros y paso de parámetros	879
Punteros y arrays	883
Memoria y datos del programa	886
Memoria dinámica	891
Punteros y datos dinámicos	895
Gestión de la memoria	907
Errores comunes	911
Arrays de datos dinámicos	916
Arrays dinámicos	928



## Tema 9 (Anexo): Punteros y memoria dinámica

---

Aritmética de punteros	940
Recorrido de arrays con punteros	953
Referencias	962
Listas enlazadas	964



# Tema 10: Introducción a la recursión

---

Concepto de recursión	983
Algoritmos recursivos	986
Funciones recursivas	987
Diseño de funciones recursivas	989
Modelo de ejecución	990
La pila del sistema	992
La pila y las llamadas a función	994
Ejecución de la función factorial()	1005
Tipos de recursión	1018
Recursión simple	1019
Recursión múltiple	1020
Recursión anidada	1022
Recursión cruzada	1026
Código del subprograma recursivo	1027
Parámetros y recursión	1032
Ejemplos de algoritmos recursivos	1034
Búsqueda binaria	1035
Torres de Hanoi	1038
Recursión frente a iteración	1043
Estructuras de datos recursivas	1045



# Apéndice: Archivos binarios

---

Flujos	1051
Archivos binarios	1054
Tamaño de los datos: El operador sizeof()	1056
Apertura de archivos binarios	1059
Lectura de archivos binarios (acceso secuencial)	1061
Escritura en archivos binarios (acceso secuencial)	1066
Acceso directo o aleatorio	1070
Ejemplos de uso de archivos binarios	1078
Ordenación de los registros del archivo	1079
Búsqueda binaria	1085
Inserción en un archivo binario ordenado	1088
Carga de los registros de un archivo en una tabla	1092
Almacenamiento de una tabla en un archivo	1093





## Referencias bibliográficas

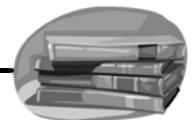
Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Referencias bibliográficas

---



- ✓ *Programming. Principles and Practice Using C++*  
B. Stroustrup. Pearson Education, 2009
- ✓ *C++: An Introduction to Computing (2ª edición)*  
J. Adams, S. Leestma, L. Nyhoff. Prentice Hall, 1998
- ✓ *El lenguaje de programación C++ (Edición especial)*  
B. Stroustrup. Addison-Wesley, 2002
- ✓ *Programación y resolución de problemas con C++*  
N. Dale, C. Weems. McGraw-Hill Interamericana, 2007
- ✓ *Problem Solving, Abstraction, Design Using C++ (3ª edición)*  
F.L. Friedman, E.B. Koffman. Addison-Wesley, 2000.
- ✓ *Programación en C++ para ingenieros*  
F. Xhafa et al. Thomson, 2006



## Referencias bibliográficas



### *Programming. Principles and Practice Using C++*

Del autor del lenguaje C++, un amplio tutorial que enseña a programar en C++; hace un uso temprano de conceptos de orientación a objetos y de la STL, que quedan fuera del temario de este curso

### *C++: An Introduction to Computing (2ª edición)*

Buena introducción a la programación en C++; buena organización de los contenidos, bien desarrollado y con secciones prácticas

### *El lenguaje de programación C++ (Edición especial)*

Del autor del lenguaje C++, la referencia absoluta sobre el lenguaje C++ en la que consultar dudas y detalles técnicos sobre los elementos del lenguaje



## Referencias bibliográficas



### *Programación y resolución de problemas con C++*

Un enfoque práctico al desarrollo de programas con C++ con numerosos ejemplos

### *Problem Solving, Abstraction, Design Using C++ (3ª edición)*

Introducción a la programación en C++ con un enfoque de desarrollo de software y numerosos casos de estudio

### *Programación en C++ para ingenieros*

Introducción a la programación en C++ con explicaciones sencillas y una organización clara





## Computadoras y programación

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Informática, computadoras y programación	3
Lenguaje máquina y ensamblador	12
Lenguajes de programación de alto nivel	15
Un poco de historia	19
Programación e Ingeniería del Software	24
El lenguaje de programación C++	27
Sintaxis de los lenguajes de programación	30
Un primer programa en C++	35
Herramientas de desarrollo	39
C++: Un mejor C	45



## Informática, computadoras y programación

Luis Hernández Yáñez



Fundamentos de la programación: Computadoras y programación

Página 3



## Informática y computadora

---

**R.A.E.**

### *Informática (Ciencia de la computación)*

Conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de ordenadores

### *Computadora*

Máquina electrónica, analógica o digital, dotada de una memoria de gran capacidad y de métodos de tratamiento de la información, capaz de resolver problemas matemáticos y lógicos mediante la ejecución de programas informáticos



Luis Hernández Yáñez



Fundamentos de la programación: Computadoras y programación

Página 4





# Computadoras

*En todas partes y con muchas formas*



Luis Hernández Yáñez



# Hardware y software

## *Hardware*

Componentes que integran la parte material de una computadora



## *Software*

Programas, instrucciones y reglas informáticas para ejecutar tareas en una computadora



Luis Hernández Yáñez



# Programación de computadoras

## Programar

Indicar a la computadora qué es lo que tiene que hacer

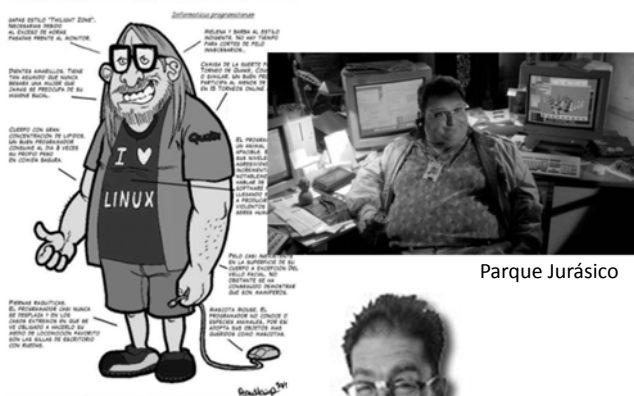
## Programa

- ✓ Secuencia de instrucciones
- ✓ Instrucciones que entiende la computadora
- ✓ Y que persiguen un objetivo: *¡resolver un problema!*



# Programadores

## ANATOMÍA DEL PROGRAMADOR GEEK



Trabajo en equipo  
Múltiples roles...

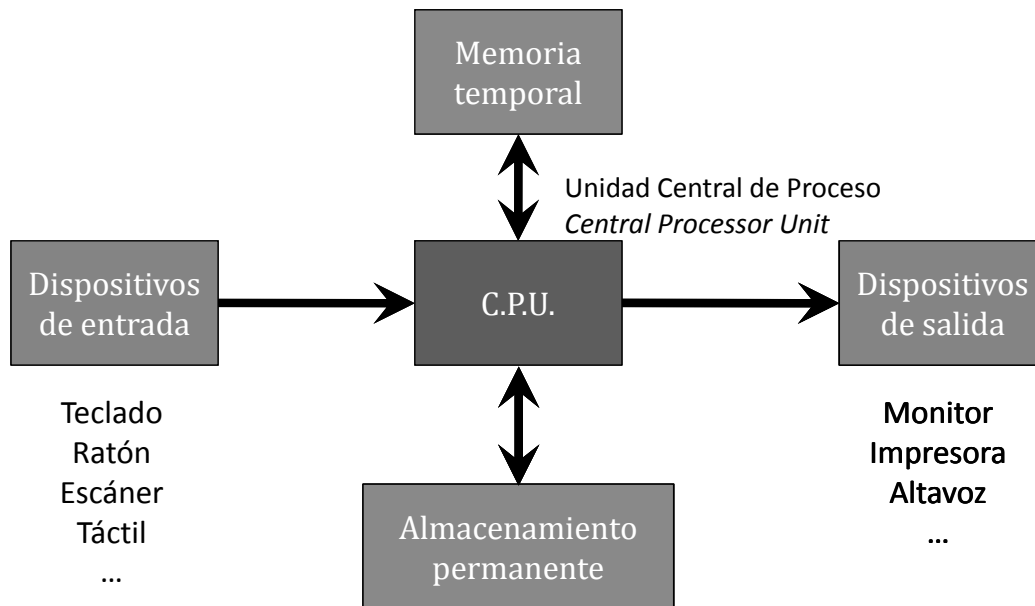
- ✓ Gestores
- ✓ Analistas
- ✓ Diseñadores
- ✓ Programadores
- ✓ Probadores
- ✓ Administradores de sistemas

...



# Computadoras

## Esquema general

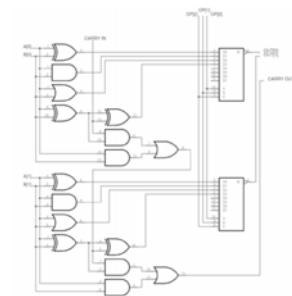
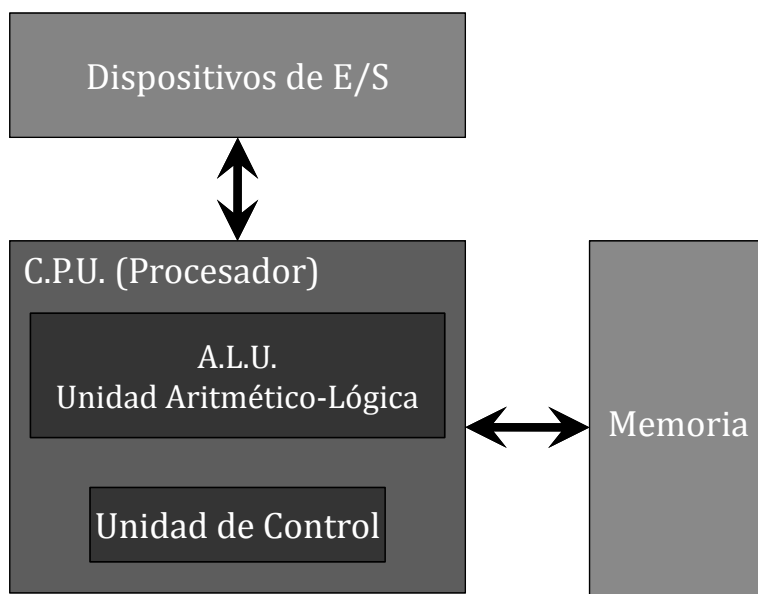


Luis Hernández Yáñez



# Computadoras

## La arquitectura de Von Neumann



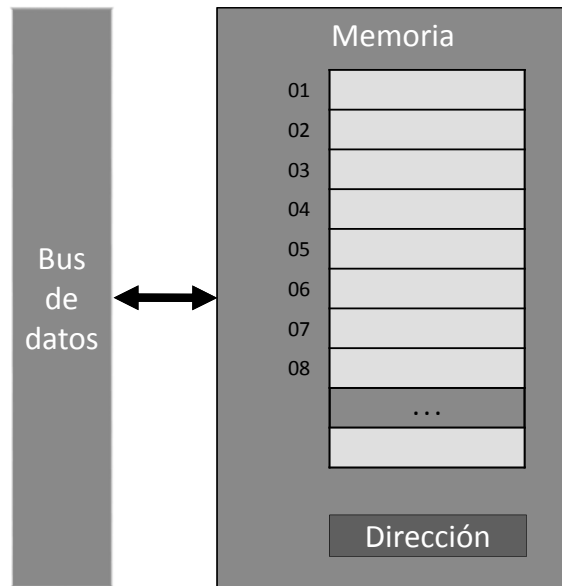
Una ALU de 2 bits (Wikipedia)

Luis Hernández Yáñez



# Computadoras

## La memoria



Cada celda en una dirección  
Celdas de 8 / 16 / 32 / 64 bits  
Información volátil

- 1 Bit = 0 / 1
- 1 Byte = 8 bits = 1 carácter
- 1 Kilobyte (KB) = 1024 Bytes
- 1 Megabyte (MB) = 1024 KB
- 1 Gigabyte (GB) = 1024 MB
- 1 Terabyte (TB) = 1024 GB
- 1 Petabyte (PB) = 1024 TB

$$2^{10} = 1024 \approx 1000$$

Luis Hernández Yáñez



# Fundamentos de la programación

## Lenguaje máquina y ensamblador

Luis Hernández Yáñez



# Programación de computadoras

*Los procesadores trabajan con ceros y unos (bits)*

Unidad de memoria básica: *Byte* (8 bits)  
(2 dígitos hexadecimales: 01011011 → 0101 1011 → 5B)

## *Lenguaje máquina*

Códigos hexadecimales que representan instrucciones, registros de la CPU, direcciones de memoria o datos

Instrucción	Significado	Lenguaje de bajo nivel
A0 2F	Acceder a la celda de memoria 2F	Dependiente de la máquina
3E 01	Copiarlo el registro 1 de la ALU	Programación difícil
A0 30	Acceder a la celda de memoria 30	
3E 02	Copiarlo en el registro 2 de la ALU	
1D	Sumar	
B3 31	Guardar el resultado en la celda de memoria 31	

Luis Hernández Yáñez



# Lenguaje ensamblador

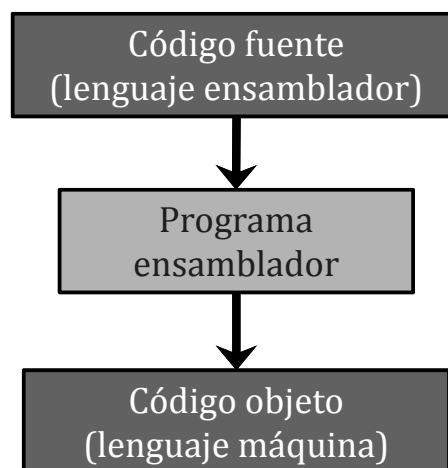
Nemotécnicos para los códigos hexadecimales:

A0 → READ    3E → REG    1D → ADD    ...

Mayor legibilidad:

READ 2F  
REG 01  
READ 30  
REG 02  
ADD  
WRITE 31

Lenguaje de nivel medio



Luis Hernández Yáñez



## Lenguajes de programación de alto nivel



## Lenguajes de programación de alto nivel

---

- ✓ Más cercanos a los lenguajes natural y matemático  
`resultado = dato1 + dato2;`
- ✓ Mayor legibilidad, mayor facilidad de codificación
- ✓ Estructuración de datos / abstracción procedimental

**FORTRAN Python Prolog C#**  
**C Pascal Cobol Lisp Ruby**  
**BASIC Smalltalk Haskell Ada**  
**Simula Java Eiffel C++**  
...

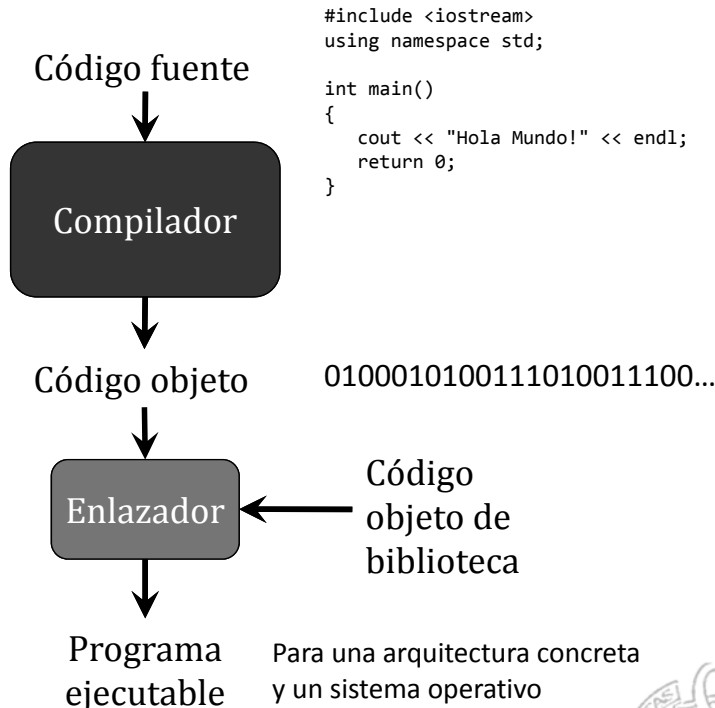


# Lenguajes de programación de alto nivel

## Traducción

**Compiladores:**  
 Compilan y enlazan programas completos

**Intérpretes:**  
 Compilan, enlazan y ejecutan instrucción a instrucción



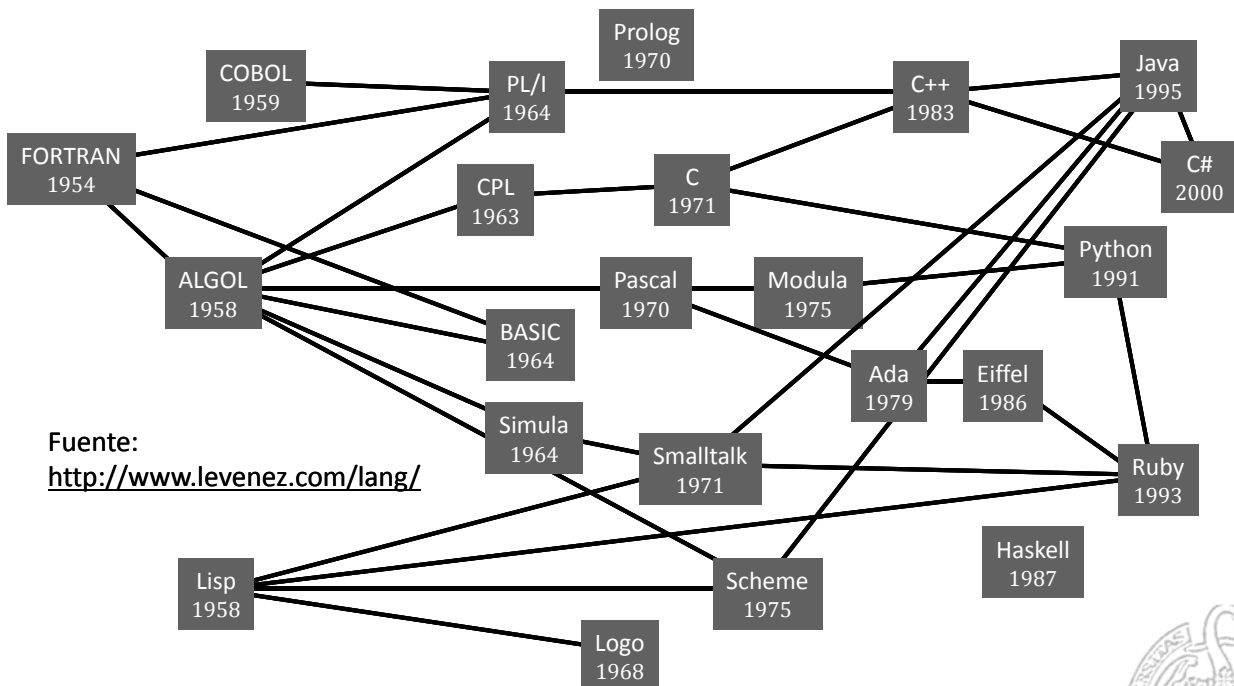
Luis Hernández Yáñez



# Los lenguajes de programación de alto nivel

## Genealogía de lenguajes

Versiones / Estándares



Fuente:  
<http://www.levenez.com/lang/>

Luis Hernández Yáñez



## Un poco de historia

Luis Hernández Yáñez



## Un poco de historia

---

### *La prehistoria*

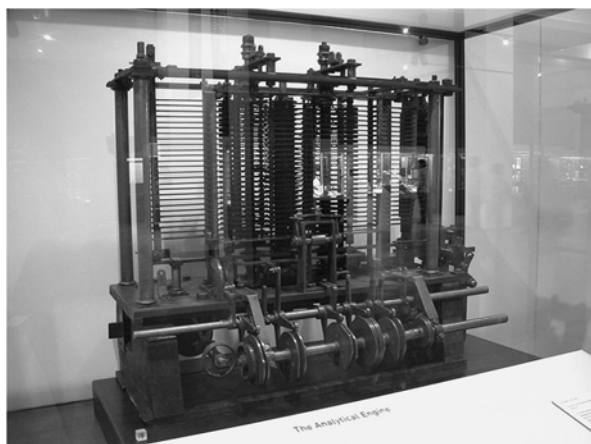
El ábaco

### Siglo XIX

Máquina analítica de Charles Babbage



(Wikipedia)



Lady Ada Lovelace es considerada la primera programadora

Luis Hernández Yáñez

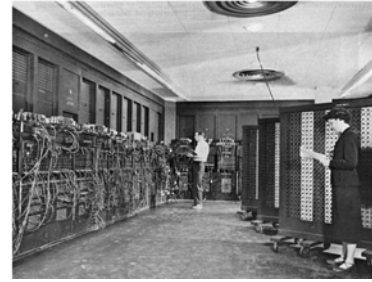




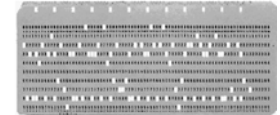
# Un poco de historia

## Siglo XX

- 1936 Máquina de Turing
- 1946 ENIAC: Primera computadora digital de propósito general
- 1947 El transistor
- 1953 IBM 650: Primera computadora a gran escala
- 1966 ARPANET: Origen de Internet
- 1967 El *disquete*
- 1970 Sistema operativo UNIX
- 1972 Primer virus informático (*Creeper*)  
Lenguaje de programación C
- 1974 Protocolo TCP. Primera red local



ENIAC (Wikipedia)



Luis Hernández Yáñez



# Un poco de historia

- 1975 Se funda Microsoft
- 1976 Se funda Apple
- 1979 Juego *Pacman*
- 1981 IBM PC  
Sistema operativo MS-DOS
- 1983 Lenguaje de programación C++
- 1984 CD-ROM
- 1985 Windows 1.0
- 1990 Lenguaje HTML  
*World Wide Web*
- 1991 Sistema operativo Linux



Apple II (Wikipedia)



IBM PC (Wikipedia)



Linux



Luis Hernández Yáñez



# Un poco de historia

---

- 1992 Windows 3.1
- 1995 Lenguaje de programación Java  
DVD
- 1998 Se funda Google
- 1999 MSN Messenger



## Siglo XXI

- 2001 Windows XP  
Mac OS X
- 2002 Mozilla Firefox
- 2007 iPhone
- 2008 Android ...



Luis Hernández Yáñez



# Fundamentos de la programación

---

## Programación e Ingeniería del Software

Luis Hernández Yáñez

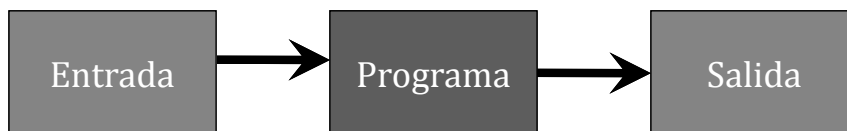


# Programa informático

## ¿Qué es programar?

Decirle a un tonto **muy rápido exactamente** lo que tiene que hacer  
Especificar la estructura y el comportamiento de un programa, así como probar que el programa realiza su tarea adecuadamente y con un rendimiento aceptable

Programa: Transforma entrada en salida



Algoritmo: Secuencia de pasos y operaciones que debe realizar el programa para resolver el problema

El programa implementa el algoritmo en un lenguaje concreto

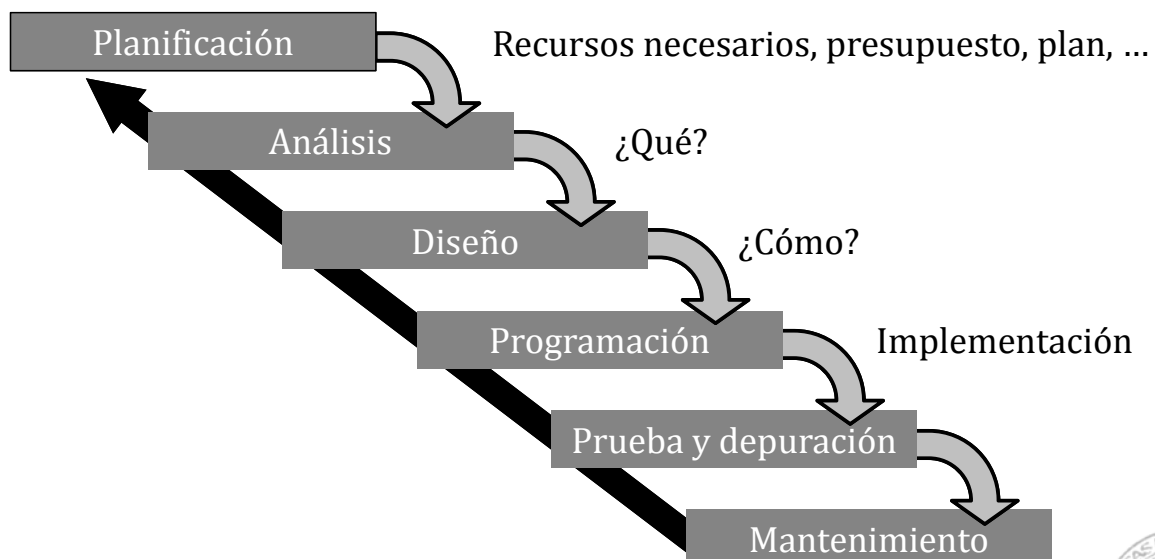
Luis Hernández Yáñez



# La Ingeniería del Software

*La programación es sólo una etapa del proceso de desarrollo*

Modelo de desarrollo “en cascada”:



Luis Hernández Yáñez



## El lenguaje de programación C++

Luis Hernández Yáñez



Fundamentos de la programación: Computadoras y programación

Página 27



## El lenguaje de programación C++

---

*Bjarne Stroustrup (1983)*

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

int main()
{
    cout << "Hola Mundo!" << endl;
    // Muestra Hola Mundo!

    return 0;
}
```

Hola Mundo!

Luis Hernández Yáñez



Fundamentos de la programación: Computadoras y programación

Página 28



# Elementos del lenguaje

---

Instrucciones

Datos: literales, variables, tipos

Subprogramas (funciones)

Comentarios

Directivas

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

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

Luis Hernández Yáñez



# Fundamentos de la programación

---

## Sintaxis de los lenguajes de programación

Luis Hernández Yáñez



# Los lenguajes de programación

## Sintaxis y semántica de los lenguajes

### Sintaxis

- Reglas que determinan cómo se pueden construir y secuenciar los elementos del lenguaje

### Semántica

- Significado de cada elemento del lenguaje  
¿Para qué sirve?



# Sintaxis de los lenguajes de programación

## Especificación

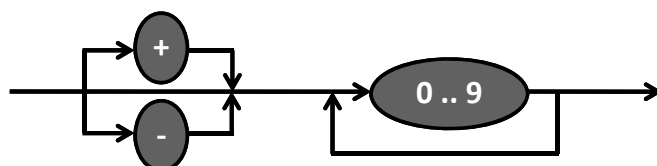
- ✓ Lenguajes (BNF)
- ✓ Diagramas

Ejemplo: Números enteros (sin decimales)

### BNF

```
<numero entero> ::= <signo opcional><secuencia de dígitos>  
<signo opcional> ::= +|-|<nada>  
<secuencia de dígitos> ::= <dígito>|<dígito><secuencia de dígitos>  
<dígito> ::= 0|1|2|3|4|5|6|7|8|9  
<nada> ::=
```

| significa ó



+23	✓
-159	✓
1374	✓
1-34	×
3.4	×
002	✓



# Backus-Naur Form (BNF)

```

<numero entero> ::= <signo opcional><secuencia de dígitos>
<signo opcional> ::= +|-|<nada>
<secuencia de dígitos> ::= <dígito>|<dígito><secuencia de dígitos>
<dígito> ::= 0|1|2|3|4|5|6|7|8|9
<nada> ::=
    
```

**+23**

```

<numero entero> ::= <signo opcional><secuencia de dígitos>
::= +<secuencia de dígitos> ::= +<dígito><secuencia de dígitos>
::= +2<secuencia de dígitos> ::= +2<dígito> ::= +23
    
```



**1374**

```

<numero entero> ::= <signo opcional><secuencia de dígitos>
::= <secuencia de dígitos> ::= <dígito><secuencia de dígitos>
::= 1<secuencia de dígitos> ::= 1<dígito><secuencia de dígitos>
::= 13<secuencia de dígitos> ::= 13<dígito><secuencia de dígitos>
::= 137<secuencia de dígitos> ::= 137<dígito> ::= 1374
    
```



**1-34**

```

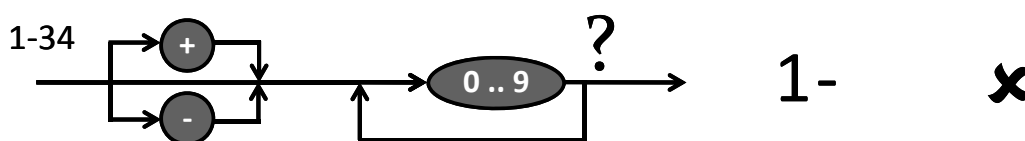
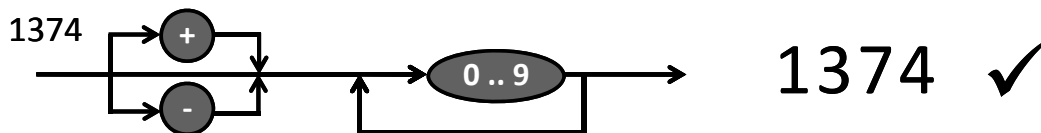
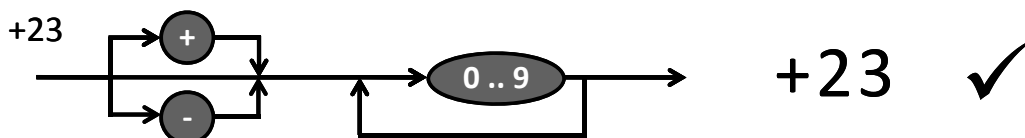
<numero entero> ::= <signo opcional><secuencia de dígitos>
::= <secuencia de dígitos> ::= <dígito><secuencia de dígitos>
::= 1<secuencia de dígitos> ::= ERROR (- no es <dígito>)
    
```



Luis Hernández Yáñez



## Diagramas de sintaxis



Luis Hernández Yáñez



## Un primer programa en C++



## Un primer programa en C++

---

*Hola Mundo!*

Un programa que muestra un saludo en la pantalla:

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

int main()
// main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl; // Muestra Hola Mundo!

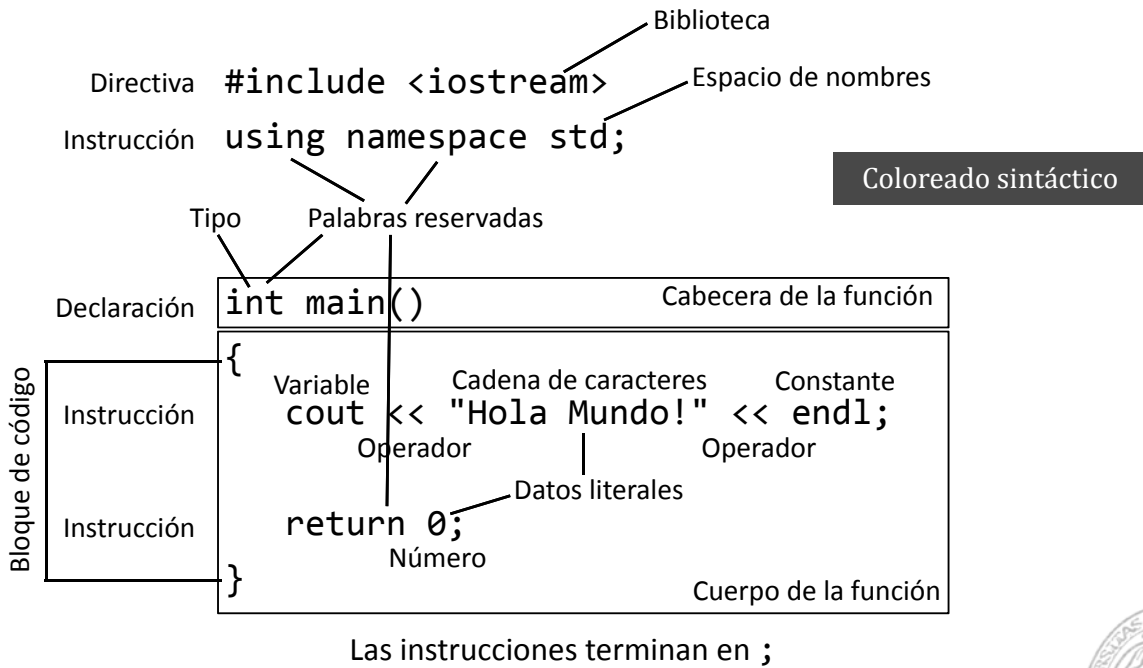
    return 0;
}
```





# Un primer programa en C++

## Análisis del programa



Luis Hernández Yáñez



# Un primer programa en C++

## Hola Mundo!

Casi todo es *infraestructura*

Sólo

```
cout << "Hola Mundo!" << endl
```

hace algo palpable

La infraestructura (notación, bibliotecas y otro soporte)  
hace nuestro código simple, completo, confiable y eficiente

*¡El estilo importa!*

Luis Hernández Yáñez



## Herramientas de desarrollo

Luis Hernández Yáñez



## Herramientas de desarrollo

---

### *Editor*

- ✓ Bloc de notas, Wordpad, Word, Writer, Gedit, Kwrite, ...  
(texto simple, sin formatos)
- ✓ Editores específicos: coloreado sintáctico
- ✓ Recomendación: Notepad++

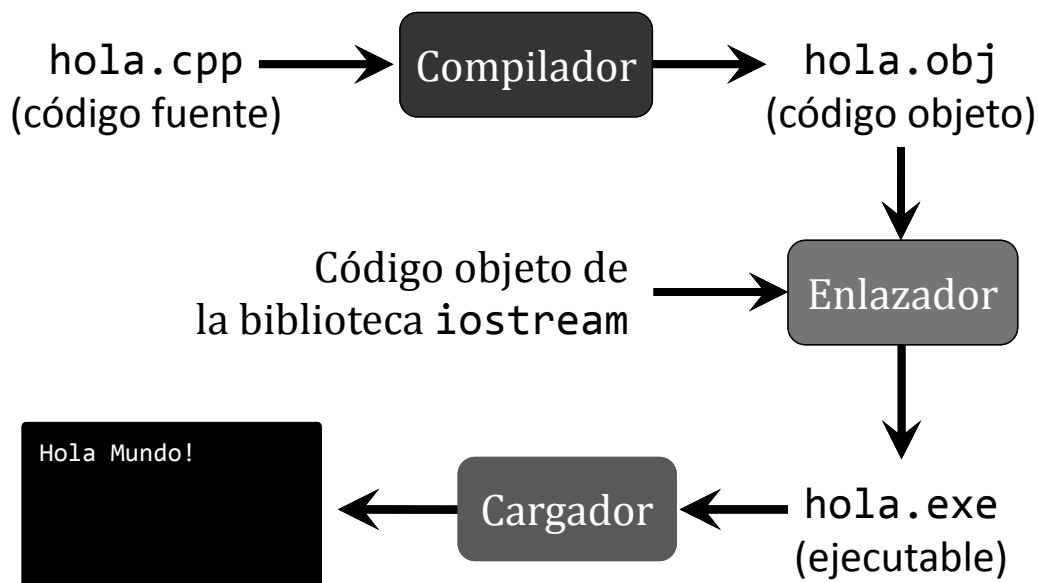
```
1 #include <iostream>
2 using namespace std;
3
4 int main() // main() es donde empieza la ejecución
5 {
6     cout << "Hola Mundo!" << endl; // Mostrar Hola Mundo!
7
8     return 0;
9 }
10
```

Instalación y uso:  
Sección  
Herramientas de desarrollo  
en el Campus Virtual

Luis Hernández Yáñez



# Compilación, enlace y ejecución



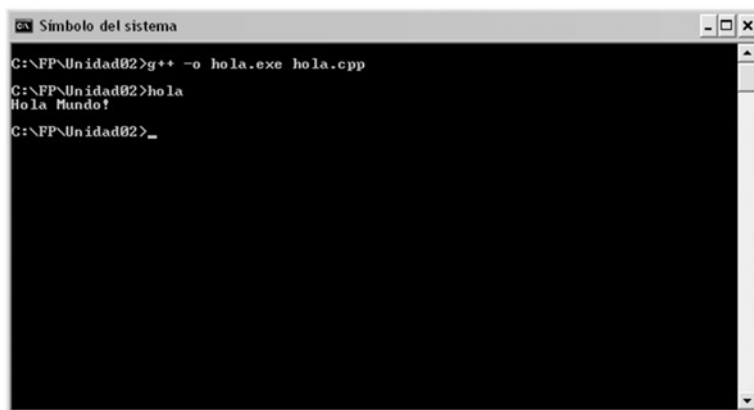
Luis Hernández Yáñez



# Más herramientas de desarrollo

## Compilador

- ✓ Importante: C++ estándar
- ✓ Recomendación: GNU G++ (*MinGW* en Windows)



Instalación y uso:  
Sección  
Herramientas de desarrollo  
en el Campus Virtual

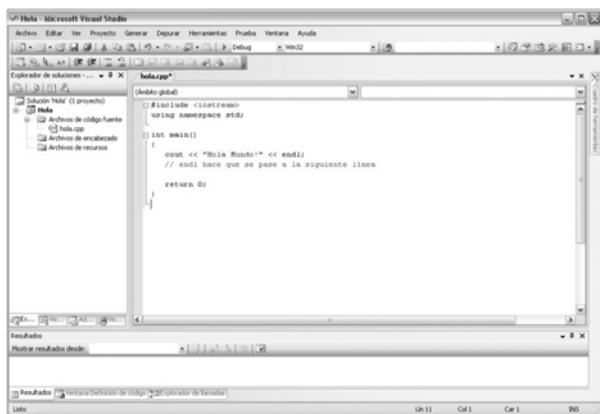
Luis Hernández Yáñez



# Más herramientas de desarrollo

## Entornos de desarrollo

- ✓ Para editar, compilar y probar el código del programa
- ✓ Recomendaciones:
  - Windows: MS Visual Studio / C++ Express o Eclipse
  - Linux: Netbeans o Eclipse



Instalación y uso:  
Sección  
Herramientas de desarrollo  
en el Campus Virtual



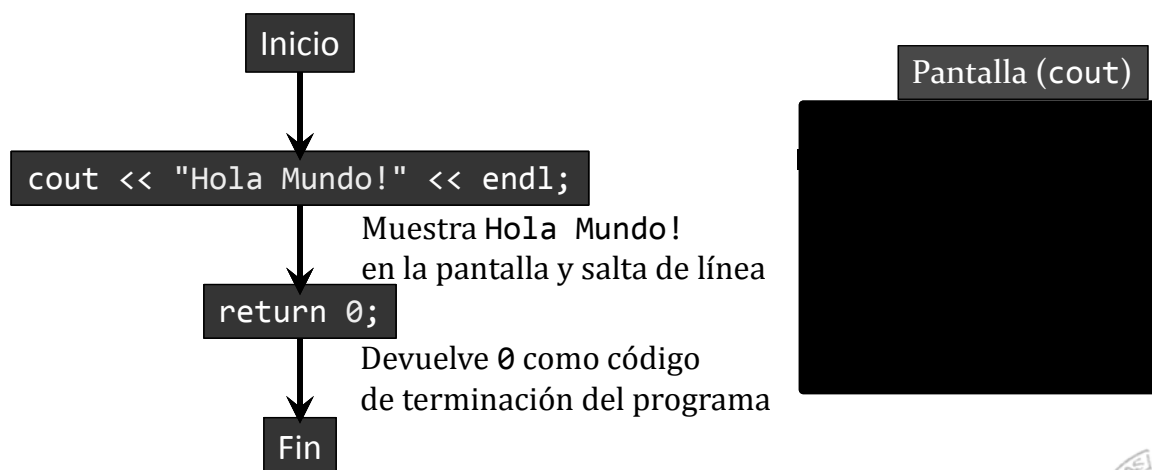
Luis Hernández Yáñez



# Un primer programa en C++: ejecución

## ¿Qué hace el programa?

- ✓ La ejecución del programa siempre empieza en `main()`
- ✓ Se ejecutan las instrucciones en secuencia de principio a fin



Luis Hernández Yáñez



## C++: Un mejor C



## C++: Un mejor C

---

### *El lenguaje C*

- ✓ Lenguaje creado por Dennis M. Ritchie en 1972
- ✓ Lenguaje de nivel medio:
  - Estructuras típicas de los lenguajes de alto nivel
  - Construcciones para control a nivel de máquina
- ✓ Lenguaje sencillo (pocas palabras reservadas)
- ✓ Lenguaje estructurado (no estrictamente estructurado en bloques)
- ✓ Compartimentalización de código (funciones) y datos (ámbitos)
- ✓ Componente estructural básico: la función (subprograma)
- ✓ Programación modular
- ✓ Distingue entre mayúsculas y minúsculas
- ✓ Palabras reservadas (o clave): en minúsculas








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.



## 2

## Tipos e instrucciones I

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Un ejemplo de programación	50	Operadores relacionales	177
El primer programa en C++	64	Toma de decisiones (if)	180
Las líneas de código del programa	80	Bloques de código	183
Cálculos en los programas	86	Bucles (while)	186
Variables	92	Entrada/salida por consola	190
Expresiones	98	Funciones definidas por el programador	199
Lectura de datos desde el teclado	108		
Resolución de problemas	119		
Los datos de los programas	127		
Identificadores	129		
Tipos de datos	133		
Declaración y uso de variables	142		
Instrucciones de asignación	147		
Operadores	152		
Más sobre expresiones	160		
Constantes	167		
La biblioteca cmath	171		
Operaciones con caracteres	174		



## Un ejemplo de programación



## Un ejemplo de programación

---

### *Una computadora de un coche*

Instrucciones que entiende:

<instrucción> ::= <inst> ;

<inst> ::= Start | Stop | <avanzar>

<avanzar> ::= Go <dirección> <num> Blocks

<dirección> ::= North | East | South | West

<num> ::= 1 | 2 | 3 | 4 | 5

Ejemplos:

Start;

Go North 3 Blocks;

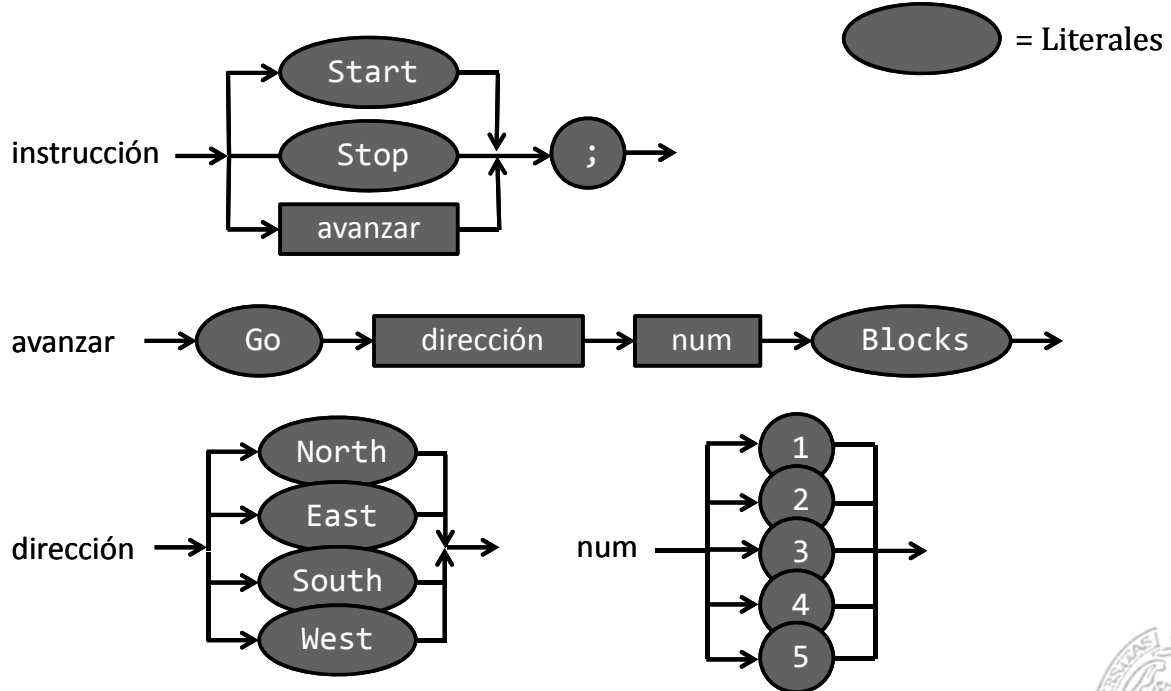
Stop;





# Un ejemplo de programación

## Sintaxis del lenguaje de programación



Luis Hernández Yáñez



# Un ejemplo de programación

## El problema a resolver

Estando el coche en la posición A, conseguir llegar al Cine Tívoli (B)

¿Qué pasos hay que seguir?

*Arrancar*

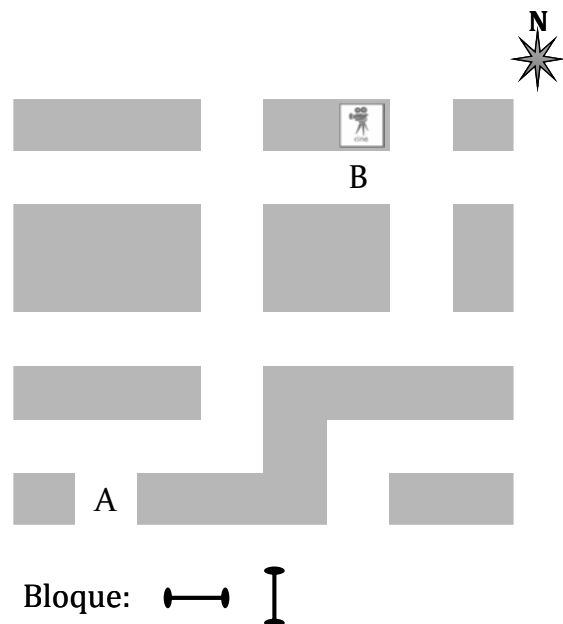
*Ir un bloque al Norte*

*Ir dos bloques al Este*

*Ir cinco bloques al Norte*

*Ir dos bloques al Este*

*Parar*



Luis Hernández Yáñez

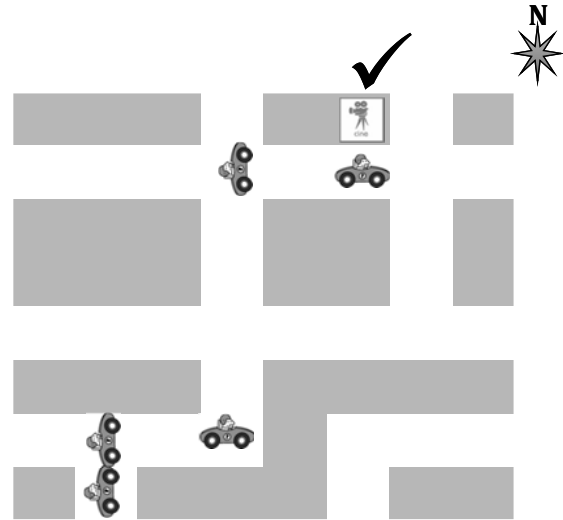


# Un ejemplo de programación

## El algoritmo

Secuencia de pasos que hay que seguir para resolver el problema

- 1.- Arrancar
- 2.- Ir un bloque al Norte
- 3.- Ir dos bloques al Este
- 4.- Ir cinco bloques al Norte
- 5.- Ir dos bloques al Este
- 6.- Parar



Esos pasos sirven tanto para una persona como para una computadora.

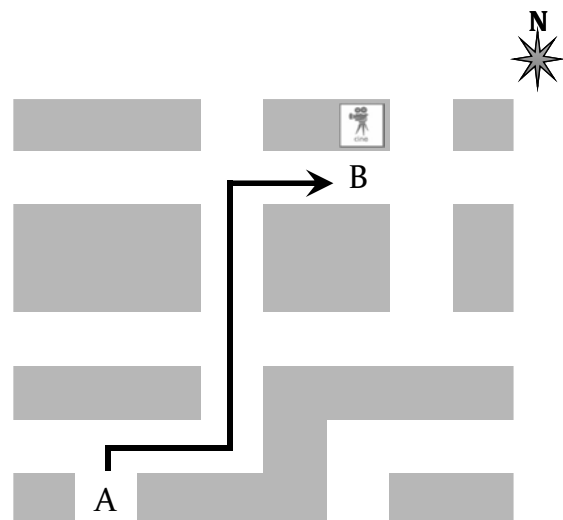


# Un ejemplo de programación

## El programa

Instrucciones escritas en el lenguaje de programación

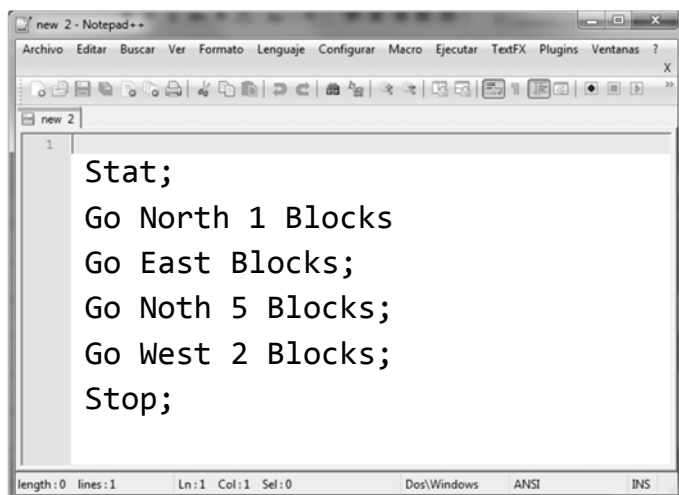
```
Start;  
Go North 1 Blocks;  
Go East 2 Blocks;  
Go North 5 Blocks;  
Go East 2 Blocks;  
Stop;
```



# Un ejemplo de programación

## El programa

Escribimos el código del programa en un editor y lo guardamos en un archivo:



```
new 2 - Notepad++
Archivo  Editar  Buscar  Ver  Formato  Lenguaje  Configurar  Macro  Ejecutar  TextFX  Plugins  Ventanas  ?
new 2
1
Stat;
Go North 1 Blocks
Go East Blocks;
Go Noth 5 Blocks;
Go West 2 Blocks;
Stop;
length:0 lines:1 Ln:1 Col:1 Sel:0 Dos\Windows ANSI INS
```

Copiamos el archivo en una llave USB y lo llevamos al coche

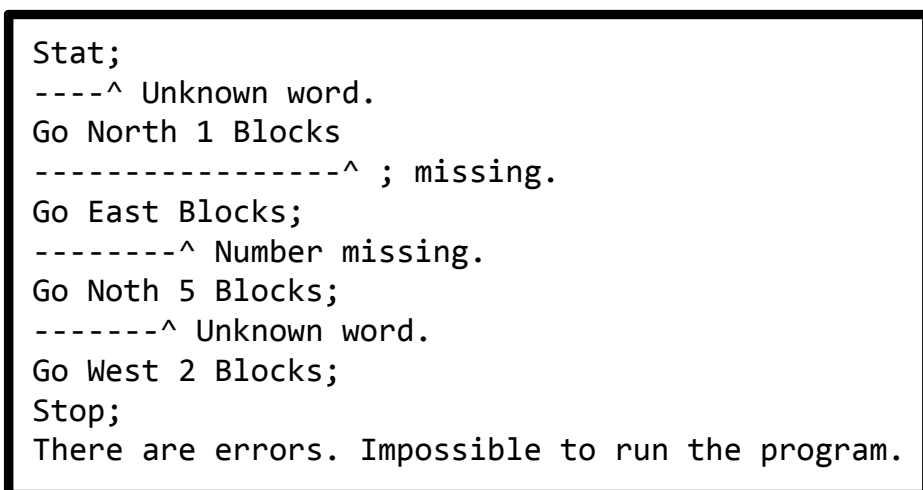
Luis Hernández Yáñez



# Un ejemplo de programación

## La compilación

Introducimos la llave USB en el coche y pulsamos el botón de ejecutar el programa:



```
Stat;
----^ Unknown word.
Go North 1 Blocks
-----^ ; missing.
Go East Blocks;
-----^ Number missing.
Go Noth 5 Blocks;
-----^ Unknown word.
Go West 2 Blocks;
Stop;
There are errors. Impossible to run the program.
```

Errores de sintaxis

Luis Hernández Yáñez



# Un ejemplo de programación

## Depuración

Editamos el código para corregir los errores sintácticos:

Stat;	→	Start;
Go North 1 Blocks		Go North 1 Blocks;
Go East Blocks;		Go East 3 Blocks;
Go Noth 5 Blocks;		Go North 5 Blocks;
Go West 2 Blocks;		Go West 2 Blocks;
Stop;		Stop;

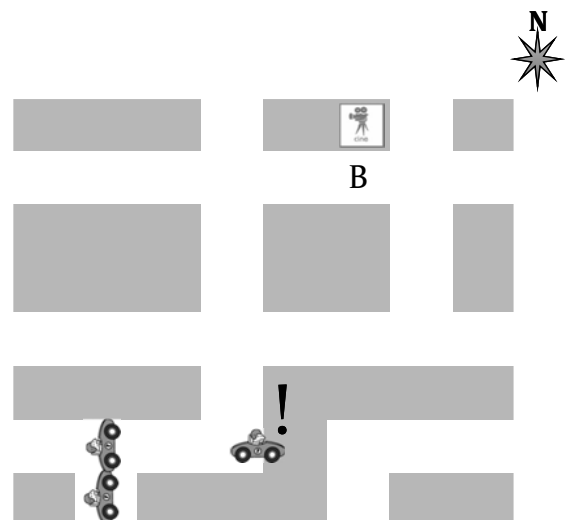


# Un ejemplo de programación

## La ejecución

Se realiza lo que pide cada instrucción:

```
Start;  
Go North 1 Blocks;  
Go East 3 Blocks;
```



Error de ejecución

*¡Una instrucción no se puede ejecutar!*



# Un ejemplo de programación

## Depuración

Editamos el código para arreglar el error de ejecución:

Start;		Start;
Go North 1 Blocks;		Go North 1 Blocks;
Go East 3 Blocks;	➔	Go East 2 Blocks;
Go North 5 Blocks;		Go North 5 Blocks;
Go West 2 Blocks;		Go West 2 Blocks;
Stop;		Stop;



# Un ejemplo de programación

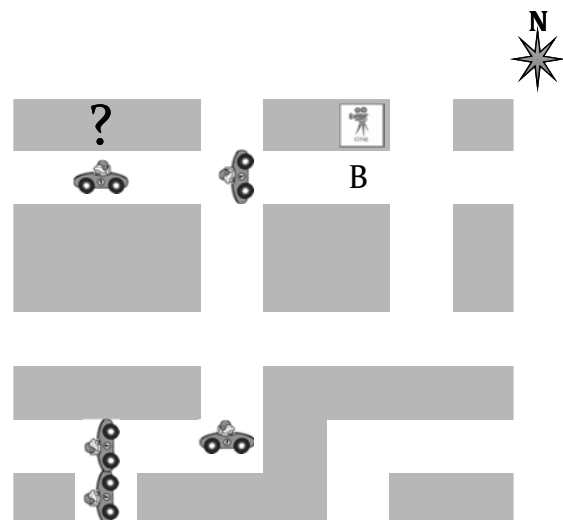
## La ejecución

Se realiza lo que pide cada instrucción:

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go West 2 Blocks;
Stop;
```

Error lógico

*¡El programa no llega al resultado deseado!*



# Un ejemplo de programación

## Depuración

Editamos el código para arreglar el error lógico:

Start;	Start;
Go North 1 Blocks;	Go North 1 Blocks;
Go East 2 Blocks;	Go East 2 Blocks;
Go North 5 Blocks;	Go North 5 Blocks;
Go West 2 Blocks;	Go East 2 Blocks;
Stop;	Stop;

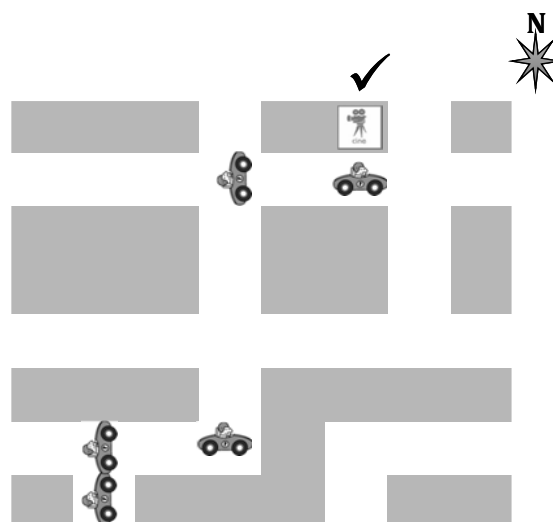


# Un ejemplo de programación

## La ejecución

Se realiza lo que pide cada instrucción:

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go East 2 Blocks;
Stop;
```



*¡Conseguido!*



## El primer programa en C++



## El primer programa en C++

---

*Hola Mundo!*

De vuelta en el programa que muestra un saludo en la pantalla:

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

int main() // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;

    return 0;
}
```



# El primer programa en C++

---

## *Hola Mundo!*

La única instrucción que produce algo tangible:

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

int main() // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;

    return 0;
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones I

Página 66



# El primer programa en C++

---

cout (iostream)

*character output stream*

Visualización en la pantalla: operador << (*insertor*)

```
cout << "Hola Mundo!" << endl;
```



```
cout << "Hola Mundo!" << endl;
```

endl → *end line*

```
Hola Mundo!
```

```
—
```

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones I

Página 67



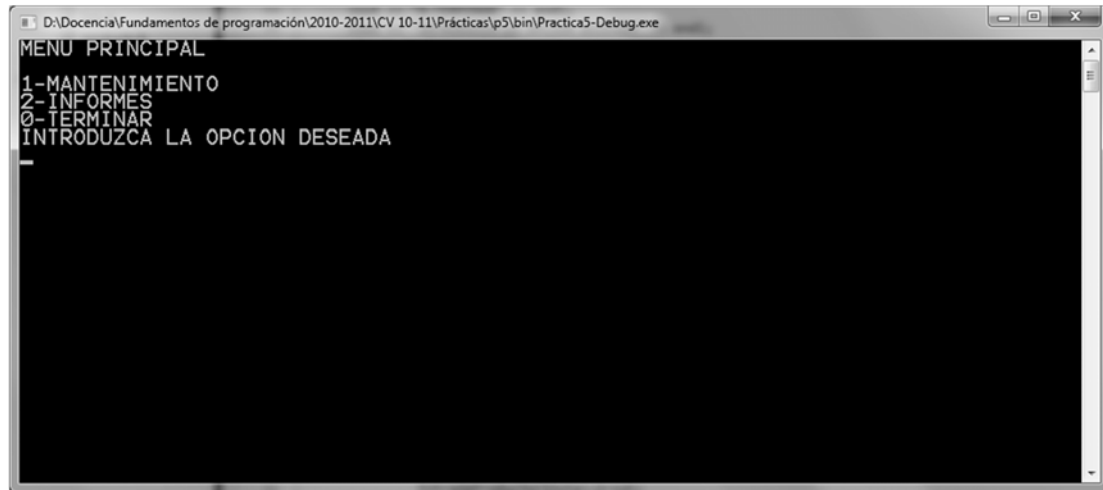


# El dispositivo de salida

## *Pantalla en modo texto*

→ Líneas de 80 caracteres (textos)

Aplicación en modo texto



Luis Hernández Yáñez

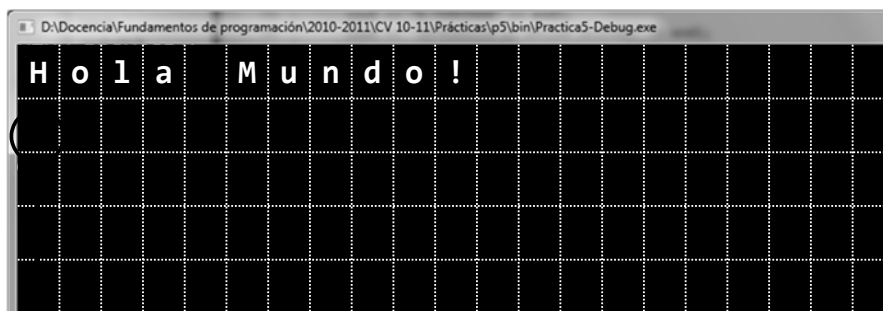


# El dispositivo de salida

## *Ventanas de consola o terminal*

Las aplicaciones en modo texto se ejecutan dentro de ventanas:

- ✓ Windows: ventanas de consola (*Símbolo del sistema*)
- ✓ Linux: ventanas de terminal



Cursor parpadeante: Donde se colocará el siguiente carácter.

Luis Hernández Yáñez



# Visualización de datos

*El insertor <<*

```
cout << ...;
```

*Inserta textos en la pantalla de modo texto*

Representación textual de los datos

A partir de la posición del cursor

*Line wrap* (continúa en la siguiente línea si no cabe)

Se pueden encadenar:

```
cout << ... << ... << ...;
```

Recuerda: las instrucciones terminan en ;



# Visualización de datos

*Con el insertor << podemos mostrar...*

- ✓ Cadenas de caracteres literales

Textos encerrados entre comillas dobles: "..."

```
cout << "Hola Mundo!";
```

*¡Las comillas no se muestran!*

- ✓ Números literales

Con o sin decimales, con signo o no: 123, -37, 3.1416, ...

```
cout << "Pi = " << 3.1416;
```

Se muestran los caracteres que representan el número

- ✓ endl

*¡Punto decimal, NO coma!*



# El primer programa en C++

## El programa principal

La función `main()`: *donde comienza la ejecución...*

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

int main() // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;
    return 0;
}
```

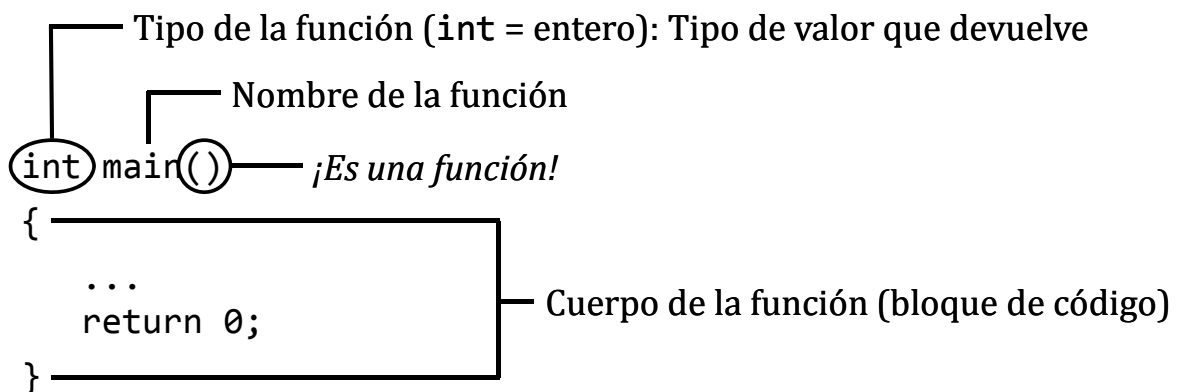
Contiene las instrucciones que hay que ejecutar



# El primer programa en C++

## El programa principal

La función `main()`:



`return 0;` Devuelve el resultado (`0`) de la función



# El primer programa en C++

---

## *Documentando el código...*

Comentarios (se ignoran):

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

```
int main()    // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;
    ...
```

Hasta el final de la línea: // Comentario de una línea

De varias líneas: /\* Comentario de varias líneas seguidas \*/



# El primer programa en C++

---

## *La infraestructura*

Código para reutilizar:

```
#include <iostream> ← Una directiva: empieza por #
using namespace std;
```

```
int main()    // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;
    return 0;
}
```

Bibliotecas de funciones a nuestra disposición



# El primer programa en C++

## Bibliotecas

Se incluyen con la *directiva* `#include`:

```
#include <iostream>
```

(Utilidades de entrada/salida por consola)

Para mostrar o leer datos hay que incluir la biblioteca `iostream`

## Espacios de nombres

En `iostream` hay espacios de nombres; ¿cuál queremos?

```
#include <iostream>
```

```
using namespace std; ← Es una instrucción: termina en ;
```

Siempre usaremos el espacio de nombres estándar (`std`)

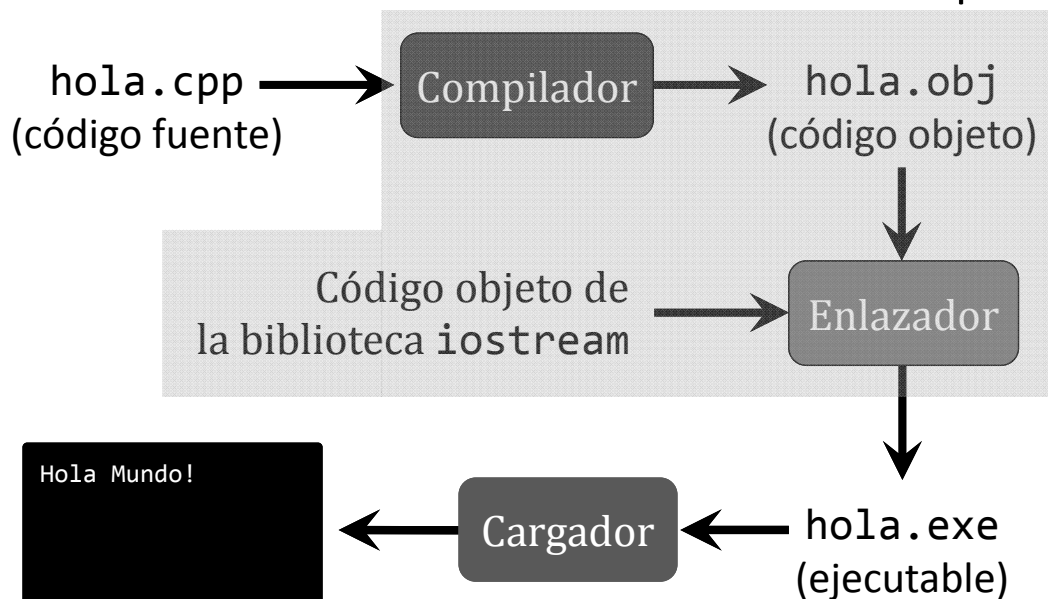
Muchas bibliotecas no tienen espacios de nombres



# El primer programa en C++

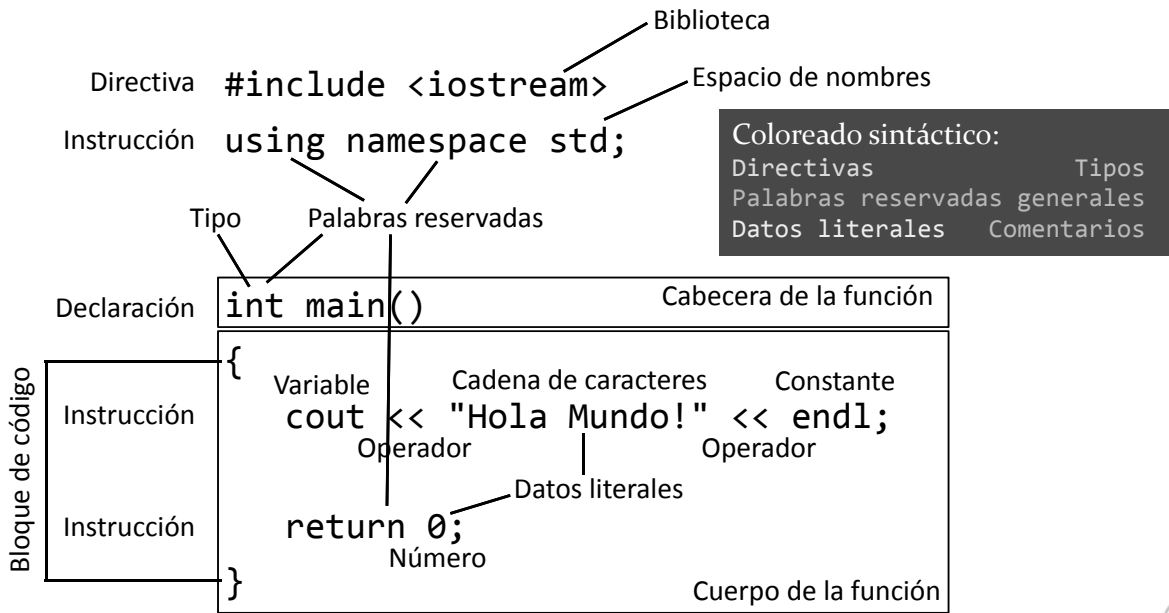
## Compilación y enlace

A menudo en un paso



# El primer programa en C++

## Elementos del programa



Las instrucciones terminan en ;

Luis Hernández Yáñez



# El primer programa en C++

## Uso de espacio en blanco

Separación de elementos por uno o más *espacios en blanco* (espacios, tabuladores y saltos de línea)

El compilador los ignora

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

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

#include <iostream> using namespace std;
int main(){cout<<"Hola Mundo!"<<endl;
return 0;}
```

¿Cuál se lee mejor?

Luis Hernández Yáñez



## Las líneas de código del programa



## Programa mínimo

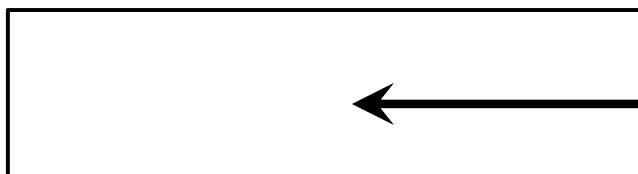
---

### *Programa con E/S por consola*

Una plantilla para empezar:

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

```
int main()
{
```



*¡Tu código aquí!*

```
return 0;
}
```





... *recitado en la consola*

Mostrar los textos con cout <<:

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

int main()
{
    cout << "En un lugar de la Mancha," << endl;
    cout << "de cuyo nombre no quiero acordarme," << endl;
    cout << "no ha mucho tiempo que vivía un hidalgo de los de
lanza en astillero, ..." << endl;
    return 0;
}
```



## Líneas de código

*Introducción del código del programa*

Terminamos cada línea de código con un salto de línea (↵):

```
#include <iostream>↵
using namespace std;↵
↵
int main()↵
{↵
    cout << "En un lugar de la Mancha," << endl;↵
    cout << "de cuyo nombre no quiero acordarme," << endl;↵
    cout << "no ha mucho tiempo que vivía un hidalgo de los de
lanza en astillero, ..." << endl;↵
    return 0;↵
}↵
```





# Líneas de código

---

## *Introducción del código del programa*

No hay que partir una cadena literal entre dos líneas:

```
cout << "no ha mucho tiempo que vivía un hidalgo de  
los de lanza en astillero, ..." << endl;
```

*¡La cadena no termina (1ª línea)!*

*¡No se entiende los (2ª línea)!*

*Veamos cómo nos muestra los errores el compilador...*



# Programar pensando en posibles cambios

---

## *Mantenimiento y reusabilidad*

- ✓ Usa espacio en blanco para separar los elementos:

```
cout << "En un lugar de la Mancha," << endl;
```

mejor que

```
cout<<"En un lugar de la Mancha,"<<endl;
```

- ✓ Usa sangría (indentación) para el código de un bloque:

```
{  
Tab → cout << "En un lugar de la Mancha," << endl;  
ó  
3 esp. | ...  
      | return 0;  
}
```

*¡El estilo importa!*



## Cálculos en los programas



## Cálculos en los programas

---

### *Operadores aritméticos*

- + Suma
- Resta
- \* Multiplicación
- / División

### Operadores binarios

*operando\_izquierdo*    *operador*    *operando\_derecho*

Operación	Resultado
3 + 4	7
2.56 - 3	-0.44
143 * 2	286
45.45 / 3	15.15



# Cálculos en los programas

## Números literales (concretos)

- ✓ Enteros: sin parte decimal

Signo negativo (opcional) + secuencia de dígitos

3 143 -12 67321 -1234

No se usan puntos de millares

- ✓ Reales: con parte decimal

Signo negativo (opcional) + secuencia de dígitos

+ punto decimal + secuencia de dígitos

3.1416 357.0 -1.333 2345.6789 -404.1



Punto decimal (3.1416), NO coma (~~3,1416~~)



# Cálculos en los programas

cálculos.cpp

## Ejemplo

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

```
int main()
{
    cout << "133 + 1234 = " << 133 + 1234 << endl;
    cout << "1234 - 111.5 = " << 1234 - 111.5 << endl;
    cout << "34 * 59 = " << 34 * 59 << endl;
    cout << "3.4 * 5.93 = " << 3.4 * 5.93 << endl;
    cout << "500 / 3 = " << 500 / 3 << endl; // Div. entera
    cout << "500.0 / 3 = " << 500.0 / 3 << endl; // Div. real

    return 0;
}
```



# Cálculos en los programas

```
D:\FP\Tema02>g++ -o cálculos cálculos.cpp
Información: se resuelve std::cout al enlazado
c:/mingw/bin/./lib/gcc/mingw32/4.5.0/./../.
importación automática se activó sin especificar
ordenes.
Esto debe funcionar a menos que involucre est
ferenciación de símbolos de DLLs auto-importadas.

D:\FP\Tema02>cálculos
133 + 1234 = 1367
1234 - 111.5 = 1122.5
34 * 59 = 2006
3.4 * 5.93 = 20.162
500 / 3 = 166
500.0 / 3 = 166.667
```

División entera

División real

Luis Hernández Yáñez



# Cálculos en los programas

## ¿División entera o división real?

Ambos operandos enteros → División entera

Algún operando real → División real

División	Resultado
500 / 3	166
500.0 / 3	166.667
500 / 3.0	166.667
500.0 / 3.0	166.667

Comprueba siempre que el tipo de división sea el que quieres

Luis Hernández Yáñez



## Variables



## Variables

---

### *Datos que se mantienen en memoria*

Variable: dato que se accede por medio de un nombre

Dato literal: un valor concreto

Variable: puede cambiar de valor (*variar*)

```
edad = 19; // variable edad y literal 19
```

Las variables deben ser declaradas

¿Qué tipo de dato queremos mantener?

- ✓ Valor numérico sin decimales (entero): tipo `int`
- ✓ Valor numérico con decimales (real): tipo `double`

Declaración: *tipo nombre;*



# Variables

## Declaración de variables

*tipo nombre;*

```
int cantidad;
```

```
double precio;
```

Se reserva espacio suficiente

cantidad

precio

Memoria

?

?

...

LAS VARIABLES NO SE INICIALIZAN

No se deben usar hasta que se les haya dado algún valor

*¿Dónde colocamos las declaraciones?*

Siempre, antes del primer uso

Habitualmente al principio de la función



# Variables

## Declaración de variables

Memoria

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int cantidad;
```

```
    double precio, total;
```

```
    return 0;
```

```
}
```

cantidad

precio

total

?

?

?

...

Podemos declarar varias de un mismo tipo separando los nombres con comas



# Variables

---

## Capacidad de las variables

int

-2.147.483.648 ... 2.147.483.647

-2147483648 .. 2147483647

double

$2,23 \times 10^{-308}$  ...  $1,79 \times 10^{+308}$  y sus negativos

[+|-] 2.23e-308 .. 1.79e+308 ← Notación científica

Problemas de precisión



# Variables

---

## Asignación de valores a las variables (operador =)

*variable* = *expresión*; ← Instrucción: termina en ;

cantidad = 12; // int

cantidad ← 12

precio = 39.95; // double

total = cantidad \* precio; // Asigna 479.4

Concordancia de tipos:

~~cantidad = 12.5;~~

*¡¡¡A la izquierda del = debe ir siempre una variable!!!*



## Expresiones



## Expresiones

---

### *Expresiones*

Secuencias de operandos y operadores

*operando operador operando operador operando ...*

```
total = cantidad * precio * 1.18;
```

Expresión

A igual prioridad se evalúan de izquierda a derecha

Paréntesis para forzar ciertas operaciones

```
total = cantidad1 + cantidad2 * precio;
```

```
total = (cantidad1 + cantidad2) * precio;
```

≠

Unos operadores se evalúan antes que otros





# Expresiones

---

## *Precedencia de los operadores*

```
cantidad1 = 10;  
cantidad2 = 2;  
precio = 40.0;
```

\* y / se evalúan antes que + y -

```
total = cantidad1 + cantidad2 * precio;
```

\* antes que + →  $10 + 2 * 40,0 \rightarrow 10 + 80,0 \rightarrow 90,0$

```
total = (cantidad1 + cantidad2) * precio;
```

+ antes que \* →  $(10 + 2) * 40,0 \rightarrow 12 * 40,0 \rightarrow 480,0$



# Variables y expresiones

---

variables.cpp

## *Ejemplo de uso de variables y expresiones*

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int cantidad;  
    double precio, total;  
    cantidad = 12;  
    precio = 39.95;  
    total = cantidad * precio;  
    cout << cantidad << " x " << precio << " = "  
        << total << endl;  
  
    return 0;  
}
```



# Variables y expresiones

## Ejemplo de uso de variables

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

int main()
{
    int cantidad;
    double precio, total;
```

	Memoria
cantidad	?
precio	?
total	?
	...



# Variables y expresiones

## Ejemplo de uso de variables

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

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
```

	Memoria
cantidad	12
precio	?
total	?
	...



# Variables y expresiones

## Ejemplo de uso de variables

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

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
```

	Memoria
cantidad	12
precio	39.95
total	?
	...



# Variables y expresiones

## Ejemplo de uso de variables

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

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
```

	Memoria
cantidad	12
precio	39.95
total	479.4
	...



# Variables y expresiones

## Ejemplo de uso de variables

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

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
    cout << cantidad << " x " << precio << " = "
         << total << endl;
}
```

Memoria

cantidad	12
precio	39.95
total	479.4

...

```
D:\FP\Tema2>variables
12 x 39.95 = 479.4
```

Luis Hernández Yáñez



# Variables y expresiones

## Ejemplo de uso de variables

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

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
    cout << cantidad << " x " << precio << " = "
         << total << endl;

    return 0;
}
```

```
Simbolo del sistema
D:\FP\Tema2>g++ -o variables variables.cpp
Información: se resuelve std::cout al enlazar acción)
c:/mingw/bin/./lib/gcc/mingw32/4.5.0/./../..
importación automática se activó sin especifica de órdenes.
Esto debe funcionar a menos que involucre estrferencién símbolos de DLLs auto-importadas.

D:\FP\Tema2>variables
12 x 39.95 = 479.4

D:\FP\Tema2>
```

Luis Hernández Yáñez



## Lectura de datos desde el teclado

Luis Hernández Yáñez



## Valores proporcionados por el usuario

---

`cin` (iostream)

*character input stream*

Lectura de valores de variables: operador `>>` (*extractor*)

```
cin >> cantidad;
```



```
cin >> cantidad;
```

Memoria

cantidad 12

...



Luis Hernández Yáñez



# Valores proporcionados por el usuario

*El extractor >>*

```
cin >> variable;
```

*Transforma los caracteres introducidos en datos*

Cursor parpadeante: lugar de lectura del siguiente carácter

La entrada termina con Intro (cursor a la siguiente línea)

*¡El destino del extractor debe ser SIEMPRE una variable!*

Se ignoran los espacios en blanco iniciales



# Valores proporcionados por el usuario

*Lectura de valores enteros (int)*

Se leen dígitos hasta encontrar un carácter que no lo sea

12abc↓    12 abc↓    12    abc↓    12↓

Se asigna el valor 12 a la variable

El resto queda pendiente para la siguiente lectura

Recomendación: Lee cada variable en una línea    12↓

*Lectura de valores reales (double)*

Se leen dígitos, el punto decimal y otros dígitos

39.95.5abc↓    39.95 abc↓    39.95↓

Se asigna el valor 39,95 a la variable; el resto queda pendiente

Recomendación: Lee cada variable en una línea    39.95↓



# Valores proporcionados por el usuario

*¿Qué pasa si el usuario se equivoca?*

El dato no será correcto

Aplicación profesional: código de comprobación y ayuda

Aquí supondremos que los usuarios no se equivocan

En ocasiones añadiremos comprobaciones sencillas



Para evitar errores, lee cada dato en una instrucción aparte



# Valores proporcionados por el usuario

*¿Qué pasa si el usuario se equivoca?*

```
int cantidad;  
double precio, total;  
cout << "Introduce la cantidad: ";  
cin >> cantidad;  
cout << "Introduce el precio: ";  
cin >> precio;  
cout << "Cantidad: " << cantidad << endl;  
cout << "Precio: " << precio << endl;
```

*¡Amigable con el usuario!  
¿Qué tiene que introducir?*

```
Introduce la cantidad: abc  
Introduce el precio: Cantidad: 0  
Precio: 1.79174e-307
```

No se puede leer un entero → 0 para cantidad y Error  
La lectura del precio falla: precio no toma valor (*basura*)



# Valores proporcionados por el usuario

*¿Qué pasa si el usuario se equivoca?*

```
Introduce la cantidad: 12abc
Introduce el precio: Cantidad: 12
Precio: 0
```

12 para cantidad  
No se puede leer un real  
→ 0 para precio y Error

```
Introduce la cantidad: 12.5abc
Introduce el precio: Cantidad: 12
Precio: 0.5
```

12 para cantidad  
.5 → 0,5 para precio  
Lo demás queda pendiente

```
Introduce la cantidad: 12
Introduce el precio: 39.95
Cantidad: 12
Precio: 39.95
```

*!!!Lectura correcta!!!*



## Programa con lectura de datos

### *División de dos números*

*Pedir al usuario dos números y mostrarle el resultado de dividir el primero entre el segundo*

Algoritmo.-

Datos / cálculos

1. Pedir el numerador

Variable numerador (double)

2. Pedir el denominador

Variable denominador (double)

3. Realizar la división, guardando el resultado

Variable resultado (double)

resultado = numerador / denominador

4. Mostrar el resultado

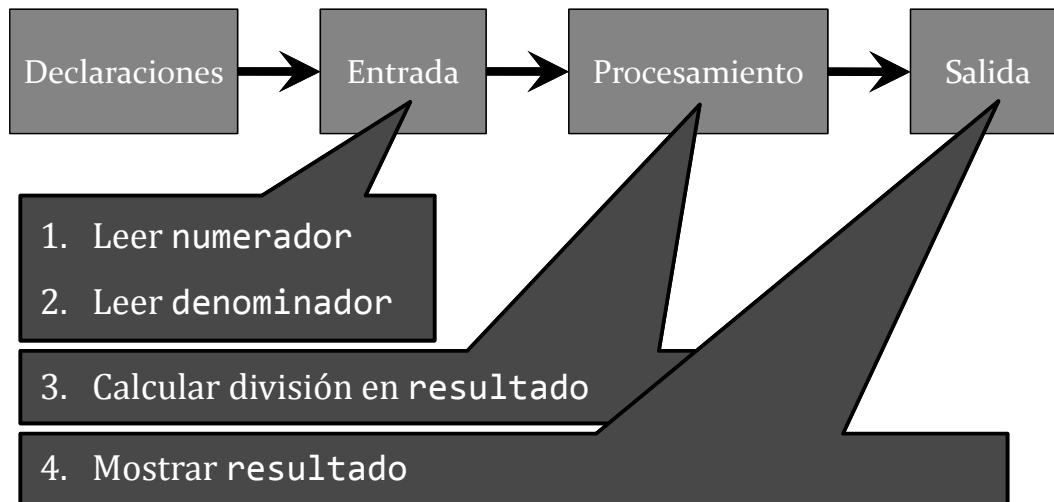




# Un esquema general

## *Entrada-Proceso-Salida*

Muchos programas se ajustan a un sencillo esquema:



Luis Hernández Yáñez



# Programa con lectura de datos

## Instrucciones

### *División de dos números*

*Pedir al usuario dos números y mostrarle el resultado de dividir el primero entre el segundo.*

1. Leer numerador

```
cin >> numerador;
```

2. Leer denominador

```
cin >> denominador;
```

3. Calcular división en resultado

```
resultado = numerador / denominador;
```

4. Mostrar resultado

```
cout << resultado;
```

Luis Hernández Yáñez



## División de dos números

división.cpp

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

```
int main()
{
```

Declaraciones	double numerador, denominador, resultado;
	cout << "Numerador: ";
Entrada	cin >> numerador;
	cout << "Denominador: ";
	cin >> denominador;
Procesamiento	resultado = numerador / denominador;
Salida	cout << "Resultado: " << resultado << endl;
	return 0;

```
129
Denominador: 2
Resultado: 64.5
```

```
}
```



# Fundamentos de la programación

## Resolución de problemas



### Problema

*Dadas la base y la altura de un triángulo, mostrar su área*

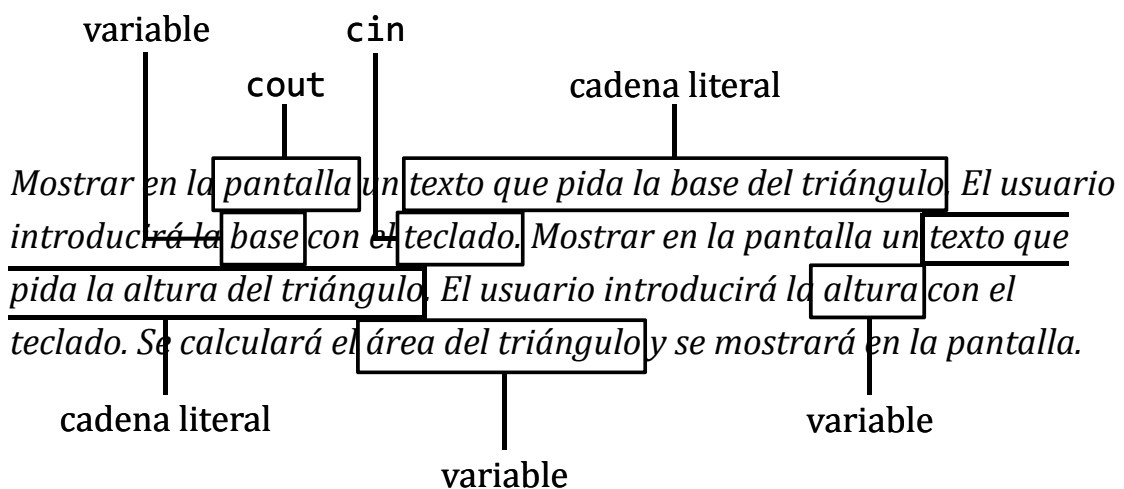


*Mostrar en la pantalla un texto que pida la base del triángulo. El usuario introducirá el valor con el teclado. Mostrar en la pantalla un texto que pida la altura del triángulo. El usuario introducirá el valor con el teclado. Se calculará el área del triángulo y se mostrará en la pantalla.*



# Resolución de problemas

## Objetos: Datos que maneja el programa



# Resolución de problemas

## Datos que maneja el programa: tipos

Objeto	Tipo	¿Varía?	Nombre
Pantalla		Variable	cout
"Introduzca la base del triángulo: "		Constante	ninguno
Base del triángulo	double	Variable	base
Teclado		Variable	cin
"Introduzca la altura del triángulo: "		Constante	ninguno
Altura del triángulo	double	Variable	altura
Área del triángulo	double	Variable	area



# Resolución de problemas

## Operaciones (acciones)

cout << ...

cin >> ...

Mostrar en la pantalla un texto que pida la base del triángulo. El usuario introducirá la base con el teclado. Mostrar en la pantalla un texto que pida la altura del triángulo. El usuario introducirá la altura con el teclado. Se calculará el área del triángulo y se mostrará en la pantalla.

area = base \* altura / 2



# El algoritmo

---

Secuencia de acciones que ha de realizar el programa para conseguir resolver el problema

1. Mostrar en la pantalla el texto que pida la base del triángulo
2. Leer del teclado el valor para la base del triángulo
3. Mostrar en la pantalla el texto que pida la altura
4. Leer del teclado el valor para la altura del triángulo
5. Calcular el área del triángulo
6. Mostrar el área del triángulo



# El programa

---

```
#include <iostream>
using namespace std;
int main()
{
```

Declaraciones

Algoritmo  
traducido  
a código  
en C++

1. Mostrar en la pantalla el texto que pida la base del triángulo
2. Leer del teclado el valor para la base del triángulo
3. Mostrar en la pantalla el texto que pida la altura del triángulo
4. Leer del teclado el valor para la altura del triángulo
5. Calcular el área del triángulo
6. Mostrar el área del triángulo

```
return 0;
```

```
}
```



## El programa: implementación

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

int main()
{
    double base, altura, area;           // Declaraciones
    cout << "Introduzca la base del triángulo: "; // 1
    cin >> base;                          // 2
    cout << "Introduzca la altura del triángulo: "; // 3
    cin >> altura;                        // 4
    area = base * altura / 2;            // 5
    cout << "El área de un triángulo de base " << base // 6
         << " y altura " << altura << " es: " << area << endl;

    return 0;
}
```

```
D:\FP\Tema02>triángulo
Introduzca la base del triángulo: 34.7
Introduzca la altura del triángulo: 12
El área de un triángulo de base 34.7 y altura 12 es: 208.2
```

¿triángulo?

 Recuerda: las instrucciones terminan en ;

Luis Hernández Yáñez



# Fundamentos de la programación

## Los datos de los programas

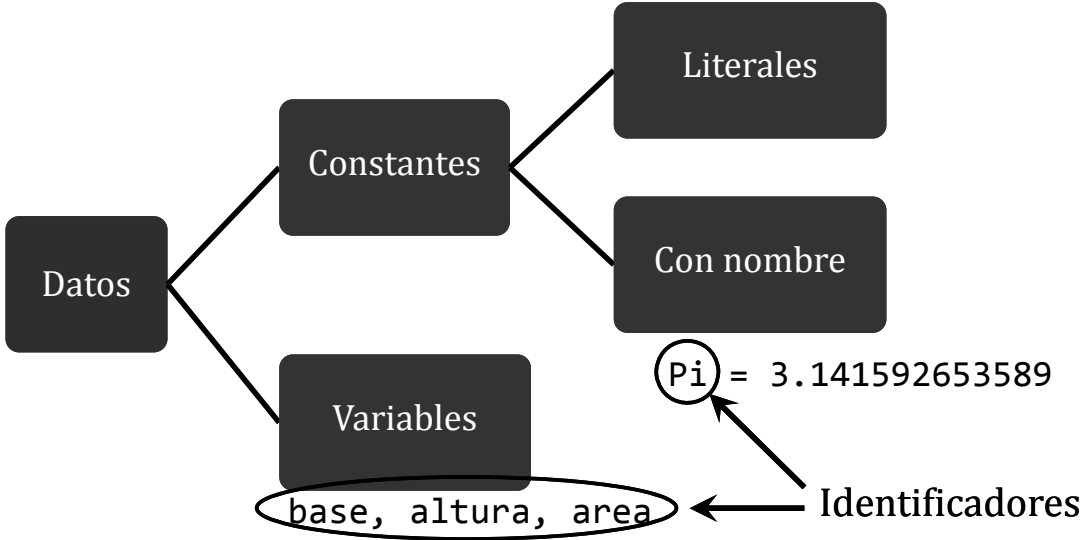
Luis Hernández Yáñez



# Los datos de los programas

## Variabilidad de los datos

"Introduzca la base del triángulo: "  
3.141592653589



Luis Hernández Yáñez



# Fundamentos de la programación

## Identificadores

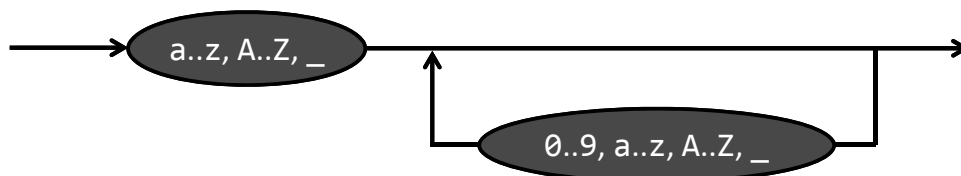
Luis Hernández Yáñez



Para variables y constantes con nombre

- *Nombre* de un dato (para accederlo/modificarlo)
- Deben ser descriptivos

Sintaxis:



cantidad prrecio total base altura area numerador

Al menos 32 caracteres significativos

👁️ *¡Ni eñes ni vocales acentuadas!*



# Identificadores

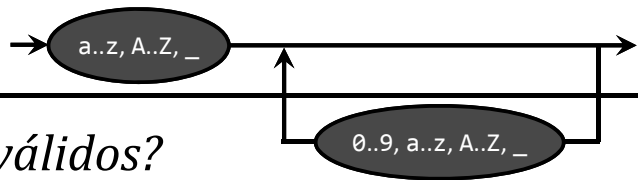
## *Palabras reservadas del lenguaje C++*

asm auto bool break case catch char class const  
const\_cast continue default delete do double  
dynamic\_cast else enum explicit extern false  
float for friend goto if inline int long  
mutable namespace new operator private protected  
public register reinterpret\_cast return short  
signed sizeof static static\_cast struct switch  
template this throw true try typedef typeid  
typename union unsigned using virtual void  
volatile while





# Identificadores



¿Qué identificadores son válidos?

balance ✓

interesAnual ✓

\_base\_imponible ✓

años ✗

EDAD12 ✓

salario\_1\_mes ✓

\_\_edad ✓

cálculoNómina ✗

valor%100 ✗

AlgunValor ✓

100caracteres ✗

valor? ✗

\_12\_meses ✓

\_\_\_valor ✓



# Fundamentos de la programación

## Tipos de datos



# Tipos de datos

## Tipos

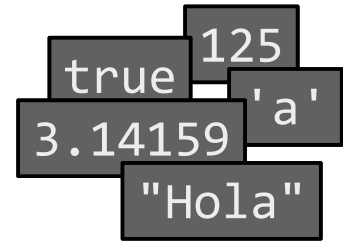
Cada dato, de un tipo concreto

Cada tipo establece:

- El conjunto (intervalo) de valores válidos
- El conjunto de operaciones que se pueden realizar

Expresiones con datos de distintos tipos (compatibles):

Transformación automática de tipos (*promoción de tipo*)



 *Anexo del Tema 2: detalles técnicos*



# Tipos de datos básicos

<code>int</code>	Números enteros (sin decimales)	1363, -12, 49	✓
<code>float</code>	Números reales	12.45, -3.1932, 1.16E+02	
<code>double</code>	Números reales (mayores intervalo y precisión)		✓
<code>char</code>	Caracteres	'a', '{', '\t'	
<code>bool</code>	Valores lógicos (verdadero/falso)	true, false	
<code>string</code>	Cadenas de caracteres (biblioteca string)	"Hola Mundo!"	
<code>void</code>	<i>Nada</i> , ausencia de tipo, ausencia de dato ( <i>funciones</i> )		



# char

## Caracteres

Intervalo de valores: Juego de caracteres (ASCII)

1 byte

Literales:

'a' '%' '\t'

Constantes de barra invertida (o *secuencias de escape*):

Caracteres de control

'\t' = tabulador '\n' = salto de línea ...

A black box containing the standard ASCII character set from code 32 to 127. The characters are arranged in five rows: Row 1: !"#\$%&amp;'()\*+,-./; Row 2: 0123456789:;&lt;=&gt;?; Row 3: @ABCDEFGHIJKLMNO; Row 4: PQRSTUVWXYZ[\]^\_`~; Row 5: abcdefghijklmno; Row 6: pqrstuvwxyz{|}~.

ASCII (códigos 32..127)

A black box containing the ISO-8859-1 character set (ASCII extended) from code 128 to 255. The characters include various Latin characters with accents, Greek letters, and other symbols.

ISO-8859-1  
(ASCII extendido: códigos 128..255)



# bool

## Valores lógicos

Sólo dos valores posibles:

- Verdadero (*true*)
- Falso (*false*)

Literales:

true false

Cualquier número distinto de 0 es equivalente a true

El 0 es equivalente a false



# Mayúsculas y minúsculas

*C++ distingue entre mayúsculas y minúsculas*

int: palabra reservada de C++ para declarar datos enteros

Int, INT o inT no son palabras reservadas de C++

true: palabra reservada de C++ para el valor *verdadero*

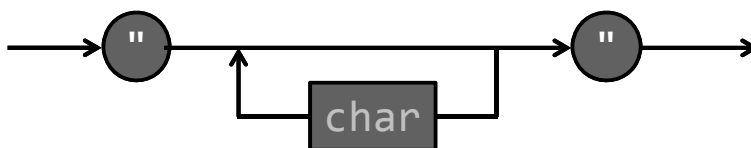
True o TRUE no son palabras reservadas de C++



## string

*Cadenas de caracteres*

"Hola" "Introduce el numerador: " "X142FG5TX?%A"



Secuencias de caracteres

Programas con variables de tipo string:

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

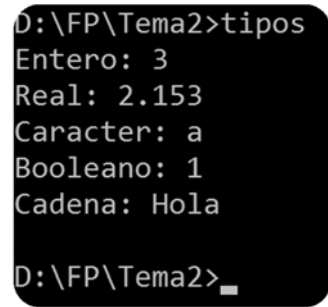
Las comillas tipográficas (apertura/cierre) “...” NO sirven  
Asegúrate de utilizar comillas rectas: "..."



```
#include <iostream>
#include <string>
using namespace std; // Un solo using... para ambas bibliotecas

int main()
{
    int entero = 3; // Podemos asignar (inicializar) al declarar
    double real = 2.153;
    char caracter = 'a';
    bool cierto = true;
    string cadena = "Hola";
    cout << "Entero: " << entero << endl;
    cout << "Real: " << real << endl;
    cout << "Carácter: " << caracter << endl;
    cout << "Booleano: " << cierto << endl;
    cout << "Cadena: " << cadena << endl;

    return 0; ¿Cuántos números hay en total en el programa?
           ¿Y caracteres? ¿Y cadenas? ¿Y booleanos?
}
```



Luis Hernández Yáñez



## Modificadores de tipos

- signed / unsigned : con signo (por defecto) / sin signo
- short / long : menor / mayor intervalo de valores

Tipo	Intervalo
int	-2147483648 .. 2147483647
unsigned int	0 .. 4294967295
short int	-32768 .. 32768
unsigned short int	0 .. 65535
long int	-2147483648 .. 2147483647
unsigned long int	0 .. 4294967295
double	+ - 2.23e-308 .. 1.79e+308
long double	+ - 3.37E-4932 .. 1.18E+4932

Luis Hernández Yáñez



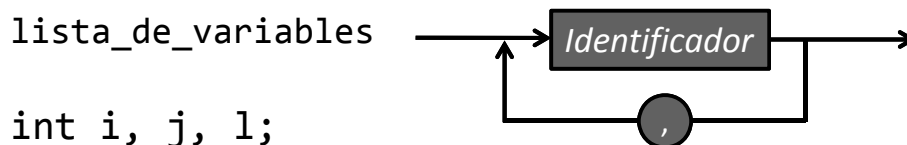
## Declaración y uso de variables



## Declaración de variables

```
[modificadores] tipo lista_de_variables;
```

└── Opcional ─┘



```
int i, j, l;  
short int unidades;  
unsigned short int monedas;  
double balance, beneficio, perdida;
```



*Programación con buen estilo:*

Identificadores descriptivos

Espacio tras cada coma

Nombres de las variables en minúsculas

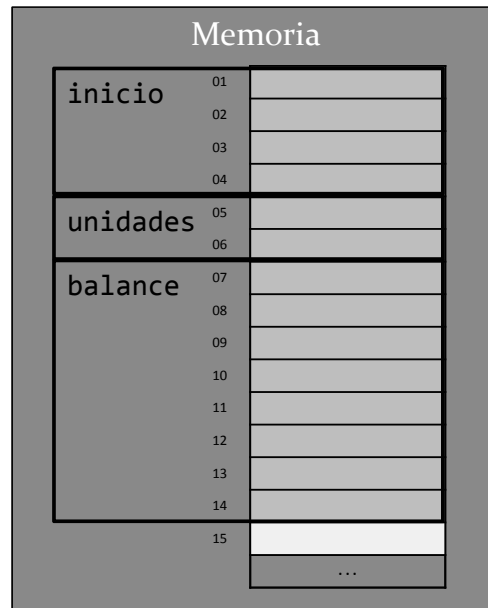
(Varias palabras: capitaliza cada inicial: `interesPorMes`)



# Datos y memoria

Se reserva memoria suficiente para cada tipo de dato

```
int inicio;  
short int unidades;  
double balance;
```



Luis Hernández Yáñez



## Inicialización de variables

*¡En C++ las variables no se inicializan automáticamente!*

*¡Una variable debe haber sido inicializada antes de ser accedida!*

¿Cómo se inicializa una variable?

- Al leer su valor (`cin >>`)
- Al asignarle un valor (instrucción de asignación)
- Al declararla

Inicialización en la propia declaración:

... → **Identificador** → = → **Expresión** → Expresión: valor compatible

```
int i = 0, j, l = 26;  
short int unidades = 100;
```

En particular, una expresión puede ser un literal

Luis Hernández Yáñez



# Uso de las variables

---

## *Obtención del valor de una variable*

- ✓ Nombre de la variable en una expresión  
`cout << balance;`  
`cout << interesPorMes * meses / 100;`

## *Modificación del valor de una variable*

- ✓ Nombre de la variable a la izquierda del =  
`balance = 1214;`  
`porcentaje = valor / 30;`

Las variables han de haber sido previamente declaradas



# Fundamentos de la programación

---

## Instrucciones de asignación





# Instrucciones de asignación

---

*El operador =*



A la izquierda, SIEMPRE una variable

```
int i, j = 2;  
i = 23 + j * 5; // i toma el valor 33
```



# Instrucciones de asignación

---

*Errores*

```
int a, b, c;
```

~~5 = a;~~

// ERROR: un literal no puede recibir un valor

~~a + 23 = 5;~~

// ERROR: no puede haber una expresión a la izda.

~~b = "abc";~~

// ERROR: un entero no puede guardar una cadena

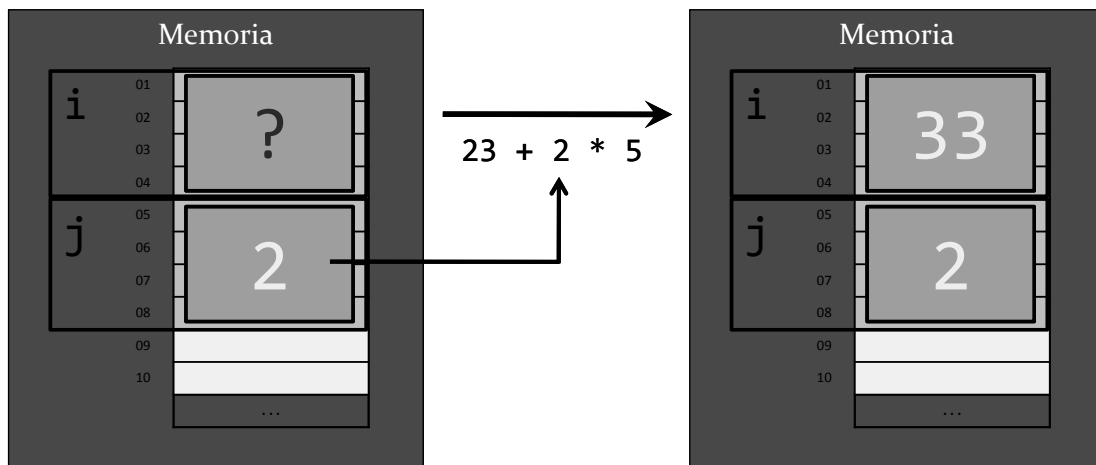
~~c = 23 5;~~

// ERROR: expresión no válida (falta operador)



# Variables, asignación y memoria

```
int i, j = 2;  
i = 23 + j * 5;
```



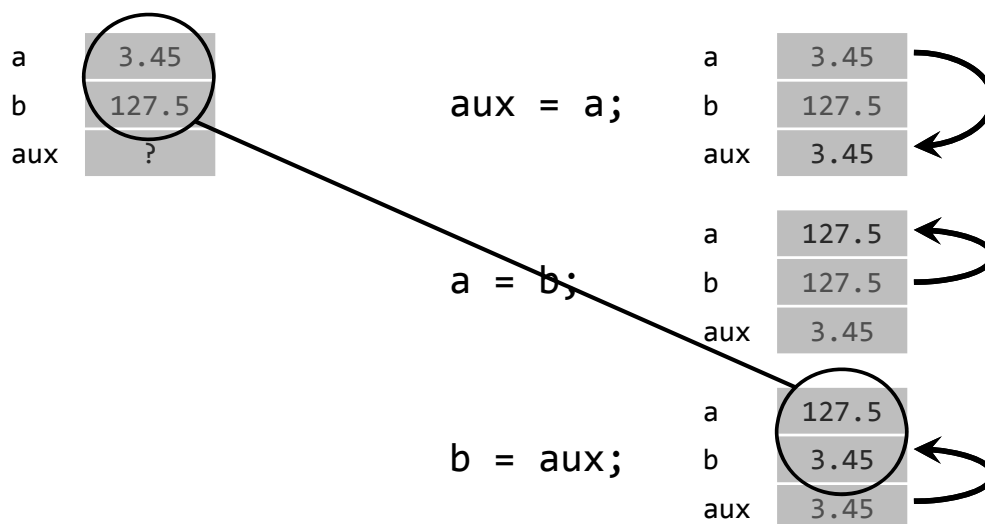
Luis Hernández Yáñez



# Ejemplo: Intercambio de valores

Necesitamos una variable auxiliar

```
double a = 3.45, b = 127.5, aux;
```



Luis Hernández Yáñez



## Operadores



## Operadores

---

### *Operaciones sobre valores de los tipos*

Cada tipo determina las operaciones posibles

Tipos de datos numéricos (`int`, `float` y `double`):

- Asignación (=)
- Operadores aritméticos
- Operadores relacionales (menor, mayor, igual, ...)

Tipo de datos `bool`:

- Asignación (=)
- Operadores lógicos (Y, O, NO)

Tipos de datos `char` y `string`:

- Asignación (=)
- Operadores relacionales (menor, mayor, igual, ...)



# Operadores aritméticos

## Operadores para tipos de datos numéricos

Operador	Operandos	Posición	int	float / double
-	1 (monario)	Prefijo	Cambio de signo	
+	2 (binario)	Infijo	Suma	
-	2 (binario)	Infijo	Resta	
*	2 (binario)	Infijo	Producto	
/	2 (binario)	Infijo	Div. entera	División real
%	2 (binario)	Infijo	Módulo	No aplicable
++	1 (monario)	Prefijo / postfijo	Incremento	
--	1 (monario)	Prefijo / postfijo	Decremento	



# Operadores aritméticos

## Operadores monarios y operadores binarios

### Operadores monarios (*unarios*)

- Cambio de signo (-):

Delante de variable, constante o expresión entre paréntesis

-saldo      -RATIO      -(3 \* a - b)

- Incremento/decremento (sólo variables) (prefijo/postfijo):

++interes      --meses      j++      // 1 más ó 1 menos

### Operadores binarios

- Operando izquierdo    operador    operando derecho

Operandos: literales, constantes, variables o expresiones

2 + 3      a \* RATIO      -a + b

(a % b) \* (c / d)



# Operadores aritméticos

*¿División entera o división real?*

/

Ambos operandos enteros: división entera

```
int i = 23, j = 2;  
cout << i / j; // Muestra 11
```

Algún operando real: división real

```
int i = 23;  
double j = 2;  
cout << i / j; // Muestra 11.5
```



# Operadores aritméticos

*Módulo (resto de la división entera)*

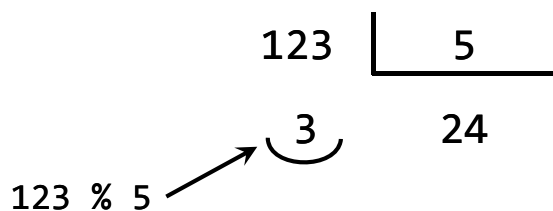
%

Ambos operandos han de ser enteros

```
int i = 123, j = 5;  
cout << i % j; // Muestra 3
```

División entera:

No se obtienen decimales → Queda un resto



# Operadores aritméticos

## Operadores de incremento y decremento

++/--

Incremento/decremento de la variable numérica en una unidad

Prefijo: Antes de acceder

```
i=i+1;
j=i;
int i = 10, j;
j = ++i; // Incrementa antes de copiar
cout << i << " - " << j; // Muestra 11 - 11
```

Postfijo: Después de acceder

```
j=i;
i=i+1;
int i = 10, j;
j = i++; // Copia y después incrementa
cout << i << " - " << j; // Muestra 11 - 10
```

👁️ No mezcles ++ y -- con otros operadores



# Operadores aritméticos: ejemplo

operadores.cpp

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

int main() {
    int entero1 = 15, entero2 = 4;
    double real1 = 15.0, real2 = 4.0;
    cout << "Operaciones entre los números 15 y 4:" << endl;
    cout << "División entera (/): " << entero1 / entero2 << endl;
    cout << "Resto de la división (%): " << entero1 % entero2 << endl;
    cout << "División real (/): " << real1 / real2 << endl;
    cout << "Num = " << real1 << endl;
    real1 = -real1;
    cout << "Cambia de signo (-): " << real1 << endl;
    real1 = -real1;
    cout << "Vuelve a cambiar (-): " << real1 << endl;
    cout << "Se incrementa antes (++ prefijo): " << ++real1 << endl;
    cout << "Se muestra antes de incrementar (posfijo ++): "
        << real1++ << endl;
    cout << "Ya incrementado: " << real1 << endl;
    return 0;
}
```



## Más sobre expresiones



## Orden de evaluación

---

*¿En qué orden se evalúan los operadores?*

$$3 + 5 * 2 / 2 - 1$$

¿De izquierda a derecha?

¿De derecha a izquierda?

¿Unos antes que otros?

Precedencia de los operadores (prioridad):

Se evalúan antes los de mayor precedencia

¿Y si tienen igual prioridad?

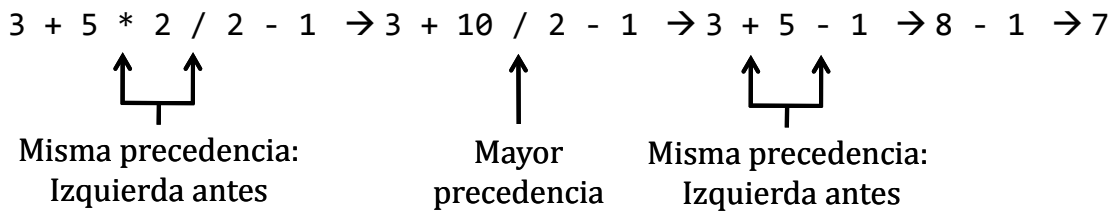
Normalmente, de izquierda a derecha

Paréntesis: fuerzan a evaluar su subexpresión



# Precedencia de los operadores

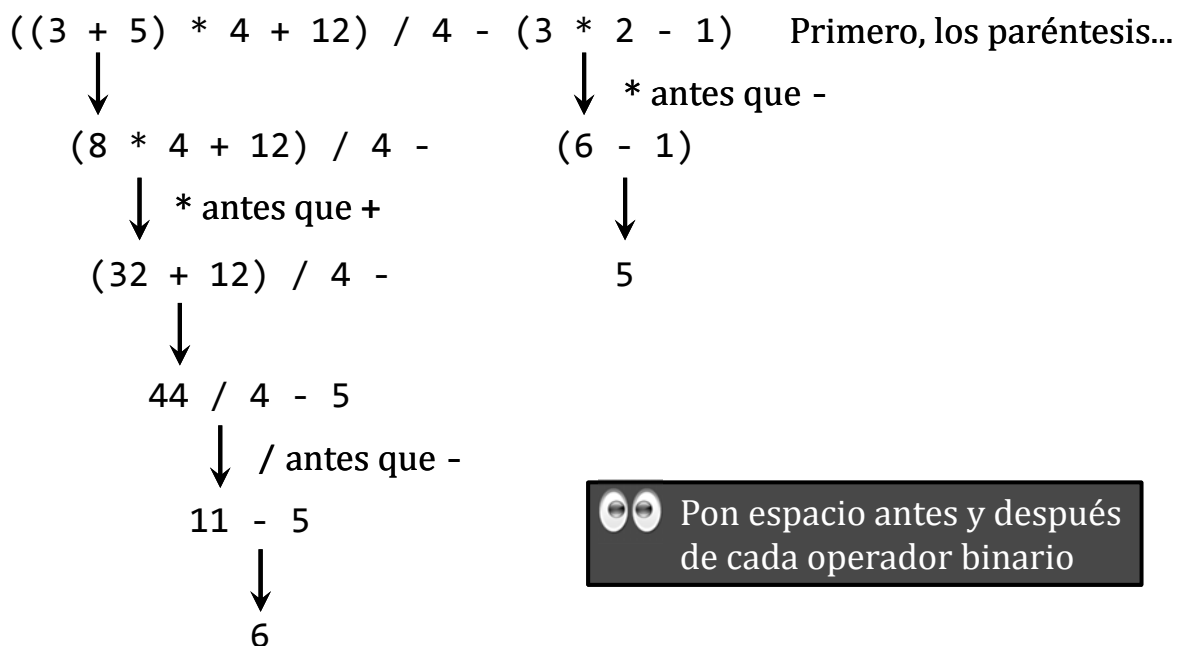
Precedencia	Operadores
Mayor prioridad	++ -- (postfijos)
	++ -- (prefijos)
	- (cambio de signo)
	* / %
Menor prioridad	+ -



Luis Hernández Yáñez



# Evaluación de expresiones



Luis Hernández Yáñez





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

int main()
{
    double x, f;
    cout << "Introduce el valor de X: ";
    cin >> x;
    f = 3 * x * x / 5 + 6 * x / 7 - 3;
    cout << "f(x) = " << f << endl;
    return 0;
}
```

$$f(x) = \frac{3x^2}{5} + \frac{6x}{7} - 3$$



Usa paréntesis para mejorar la legibilidad:

```
f = (3 * x * x / 5) + (6 * x / 7) - 3;
```



## Abreviaturas aritméticas

*variable* = ~~*variable*~~ *operador op\_derecho*;  
↑ La misma ↑ ≡  
*variable operador* = *op\_derecho*;

### Asignación

*a* = *a* + 12;

*a* = *a* \* 3;

*a* = *a* - 5;

*a* = *a* / 37;

*a* = *a* % *b*;

### Abreviatura

*a* += 12;

*a* \*= 3;

*a* -= 5;

*a* /= 37;

*a* %= *b*;

Igual precedencia  
que la asignación

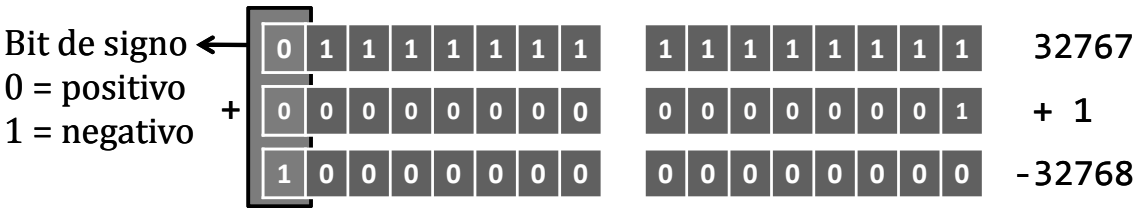
De momento,  
mejor evitarlas



# Desbordamiento

¿Valor siguiente al máximo?

```
Valor mayor del máximo (o menor del mínimo) del tipo
short int i = 32767; // Valor máximo para short int
i++; // 32768 no cabe en un short int
cout << i; // Muestra -32768
```



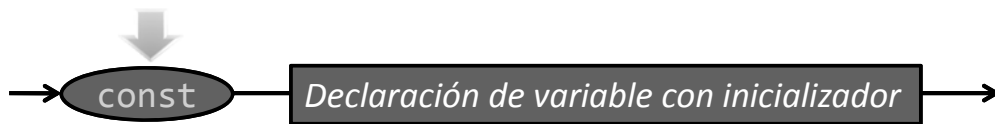
# Fundamentos de la programación

## Constantes



# Constantes

*Declaración de constantes*    Modificador de acceso `const`  
*Variables inicializadas a las que no dejamos variar*



```
const short int Meses = 12;  
const double Pi = 3.141592,  
RATIO = 2.179 * Pi;
```

La constante no podrá volver a aparecer a la izquierda de un =



*Programación con buen estilo:*  
Pon en mayúscula la primera letra de una constante o todo su nombre



## ¿Por qué utilizar constantes con nombre?

- ✓ Aumentan la legibilidad del código

```
cambioPoblacion = (0.1758 - 0.1257) * poblacion;        VS.  
cambioPoblacion = (RatioNacimientos - RatioMuertes) * poblacion;
```

- ✓ Facilitan la modificación del código

```
double compra1 = bruto1 * 18 / 100;  
double compra2 = bruto2 * 18 / 100;  
double total = compra1 + compra2;  
cout << total << " (IVA: " << 18 << "%)" << endl;  
  
const int IVA = 18;  
double compra1 = bruto1 * IVA / 100;  
double compra2 = bruto2 * IVA / 100;  
double total = compra1 + compra2;  
cout << total << " (IVA: " << IVA << "%)" << endl;
```

3 cambios ←

¿Cambio del IVA al 21%?

1 cambio ←



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

int main() {
    const double Pi = 3.141592;
    double radio = 12.2, circunferencia;
    circunferencia = 2 * Pi * radio;
    cout << "Circunferencia de un círculo de radio "
         << radio << ": " << circunferencia << endl;
    const double Euler = 2.718281828459; // Número e
    cout << "Número e al cuadrado: " << Euler * Euler << endl;
    const int IVA = 21;
    int cantidad = 12;
    double precio = 39.95, neto, porIVA, total;
    neto = cantidad * precio;
    porIVA = neto * IVA / 100;
    total = neto + porIVA;
    cout << "Total compra: " << total << endl;
    return 0;
}
```



## Fundamentos de la programación

# La biblioteca cmath



Algunas ...	<code>abs(x)</code>	Valor absoluto de x
	<code>pow(x, y)</code>	x elevado a y
	<code>sqrt(x)</code>	Raíz cuadrada de x
	<code>ceil(x)</code>	Menor entero que es mayor o igual que x
	<code>floor(x)</code>	Mayor entero que es menor o igual que x
	<code>exp(x)</code>	$e^x$
	<code>log(x)</code>	Ln x (logaritmo natural de x)
	<code>log10(x)</code>	Logaritmo en base 10 de x
	<code>sin(x)</code>	Seno de x
	<code>cos(x)</code>	Coseno de x
	<code>tan(x)</code>	Tangente de x
	<code>round(x)</code>	Redondeo al entero más próximo
	<code>trunc(x)</code>	Pérdida de la parte decimal (entero)



## La biblioteca `cmath`

`mates.cpp`

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

$$f(x, y) = 2x^5 + \frac{\sqrt{x^3}}{|x \times y|} - \cos(y)$$

```
int main() {
    double x, y, f;
    cout << "Valor de X: ";
    cin >> x;
    cout << "Valor de Y: ";
    cin >> y;
    f = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))
        / abs(x * y) - cos(y);
    cout << "f(x, y) = " << f << endl;
    return 0;
}
```

`pow()` con argumento entero:  
Usa el molde `double()`:  
`pow(double(i), 5)`



Pon un espacio detrás de cada coma en las listas de argumentos



## Operaciones con caracteres



## Operaciones con caracteres

---

char

Asignación, ++/-- y operadores relacionales

*Funciones para caracteres (biblioteca ctype)*

`isalnum(c)` true si `c` es una letra o un dígito

`isalpha(c)` true si `c` es una letra

`isdigit(c)` true si `c` es un dígito

`islower(c)` true si `c` es una letra minúscula

`isupper(c)` true si `c` es una letra mayúscula

false en caso contrario

`toupper(c)` devuelve la mayúscula de `c`

`tolower(c)` devuelve la minúscula de `c`

...



```
...
#include <cctype>

int main() {
    char caracter1 = 'A', caracter2 = '1', caracter3 = '&';
    cout << "Carácter 1 (" << caracter1 << ").-" << endl;
    cout << "Alfanumérico? " << isalnum(caracter1) << endl;
    cout << "Alfabético? " << isalpha(caracter1) << endl;
    cout << "Dígito? " << isdigit(caracter1) << endl;
    cout << "Mayúscula? " << isupper(caracter1) << endl;
    caracter1 = tolower(caracter1);
    cout << "En minúscula: " << caracter1 << endl;
    cout << "Carácter 2 (" << caracter2 << ").-" << endl;
    cout << "Alfabético? " << isalpha(caracter2) << endl;
    cout << "Dígito? " << isdigit(caracter2) << endl;
    cout << "Carácter 3 (" << caracter3 << ").-" << endl;
    cout << "Alfanumérico? " << isalnum(caracter3) << endl;
    cout << "Alfabético? " << isalpha(caracter3) << endl;
    cout << "Dígito? " << isdigit(caracter3) << endl;
    return 0;
}
```

1 ≡ true / 0 ≡ false

Luis Hernández Yáñez



## Fundamentos de la programación

# Operadores relacionales (condiciones simples)

Luis Hernández Yáñez



# Expresiones lógicas (*booleanas*)

## Operadores relacionales

Comparaciones (*condiciones*)

Condición simple ::= Expresión Operador\_relacional Expresión

Concordancia de tipo entre las expresiones

Resultado: `bool` (`true` o `false`)

<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual que
!=	distinto de

### Operadores (prioridad)

...
* / %
+ -
< <= > >=
== !=
= += -= *= /= %=



## Operadores relacionales

Menor prioridad que los operadores aditivos y multiplicativos

`bool resultado;`

```
int a = 2, b = 3, c = 4;
```

```
resultado = a < 5; // 2 < 5 → true
```

```
resultado = a * b + c >= 12; // 10 >= 12 → false
```

```
resultado = a * (b + c) >= 12; // 14 >= 12 → true
```

```
resultado = a != b; // 2 != 3 → true
```

```
resultado = a * b > c + 5; // 6 > 9 → false
```

```
resultado = a + b == c + 1; // 5 == 5 → true
```



No confundas el operador de igualdad (`==`) con el operador de asignación (`=`)



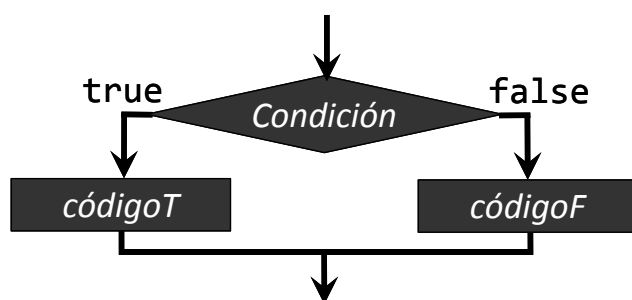


## Toma de decisiones (if)



### Hacer esto... o hacer esto otro...

*Selección: bifurcación condicional*



```
if (condición) {  
    ↪ códigoT  
}  
else {  
    ↪ códigoF  
}
```

Opcional: puede no haber else

```
int num;  
cout << "Número: ";  
cin >> num;  
if (num % 2 == 0) {  
    cout << num << " es par";  
}  
else {  
    cout << num << " es impar";  
}
```



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

int main() {
    int op1 = 13, op2 = 4;
    int opcion;
    cout << "1 - Sumar" << endl;
    cout << "2 - Restar" << endl;
    cout << "Opción: ";
    cin >> opcion;
    if (opcion == 1) {
        cout << op1 + op2 << endl;
    }
    else {
        cout << op1 - op2 << endl;
    }
    return 0;
}
```

```
D:\FP\Tema02>selección
1 - Sumar
2 - Restar
Opción: 1
17

D:\FP\Tema02>selección
1 - Sumar
2 - Restar
Opción: 2
9
```



## Fundamentos de la programación

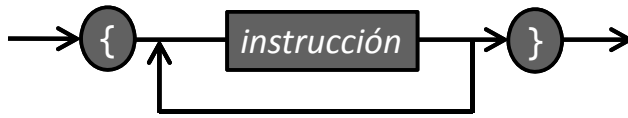
# Bloques de código



# Bloques de código

## Agrupación de instrucciones

Grupo de instrucciones a ejecutar en una rama del if



```
int num, total = 0;
cin >> num;
if (num > 0)
```

```
{
    cout << "Positivo";
    total = total + num;
}
cout << endl;
```

```
{
  | instrucción1
  | instrucción2
  | ...
  | instrucciónN
}
```

Tab ó  
3 esp.

Ámbito local  
(declaraciones locales)

Luis Hernández Yáñez



# Bloques de código

## Posición de las llaves: cuestión de estilo

```
if (num > 0)
{
    cout << "Positivo";
    total = total + num;
}
cout << endl;
```

```
if (num > 0) {
    cout << "Positivo";
    total = total + num;
}
cout << endl;
```

## No necesitamos las llaves si sólo hay una instrucción

```
if (num > 0) {
    cout << "Positivo";
}
≡
if (num > 0)
    cout << "Positivo";
```

Usaremos siempre llaves por simplicidad...

Evita poner el if y la instrucción objetivo en la misma línea:

```
if (num > 0) cout << "Positivo"; 
```

Luis Hernández Yáñez

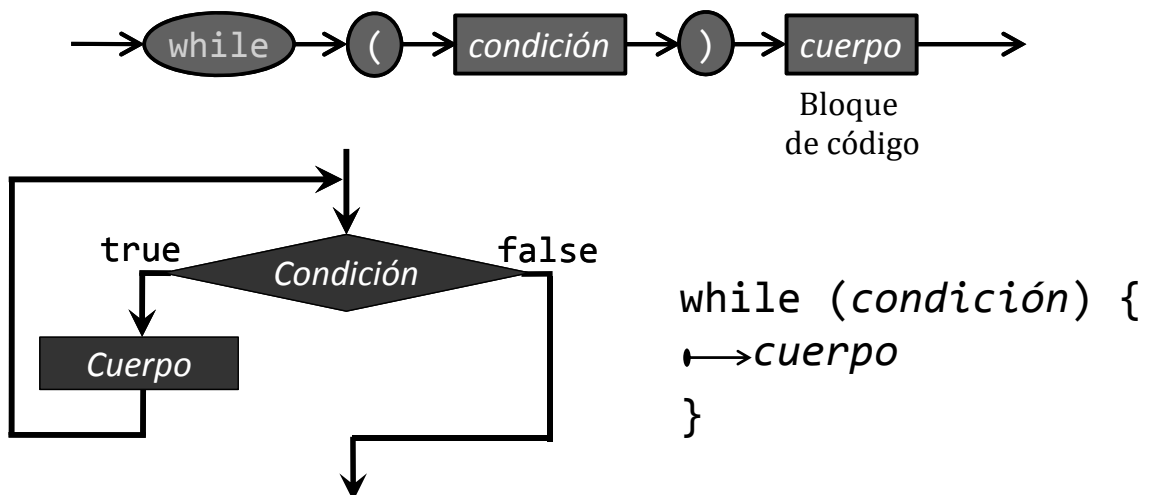


## Bucles (while)



### ***Mientras la condición sea cierta, repetir...***

*Repetición o iteración condicional*



👁️ Si la condición es false al empezar, no se ejecuta el cuerpo ninguna vez

# La instrucción while

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

int main() {
    int i = 1, n = 0, suma = 0;
    while (n <= 0) { // Sólo n positivo
        cout << "¿Cuántos números quieres sumar? ";
        cin >> n;
    }
    while (i <= n) {
        suma = suma + i;
        i++;
    }
    cout << "Sumatorio de i (1 a " << n << ") = "
        << suma << endl;
    return 0;
}
```

$$\sum_{i=1}^n i$$

```
D:\FP\Tema02>serie
¿Cuántos números quieres sumar? -3
¿Cuántos números quieres sumar? 0
¿Cuántos números quieres sumar? 5
Sumatorio de i (1 a 5) = 15
```

Luis Hernández Yáñez

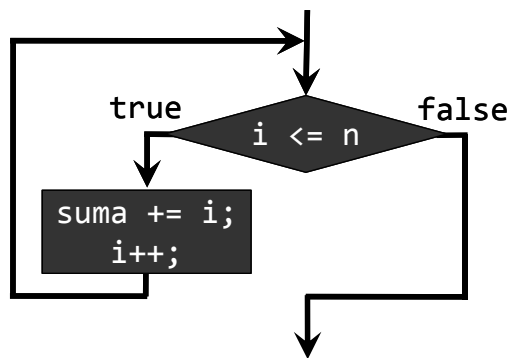


# La instrucción while

## Iteración condicional

```
while (i <= n) {
    suma = suma + i;
    i++;
}
```

$$\sum_{i=1}^n i$$



n	5
i	6
suma	15

Sumatorio de i (1 a 5) = 15

Luis Hernández Yáñez



## Entrada/salida por consola



## Entrada/salida por consola (teclado/pantalla)

*Flujos de texto (streams)*

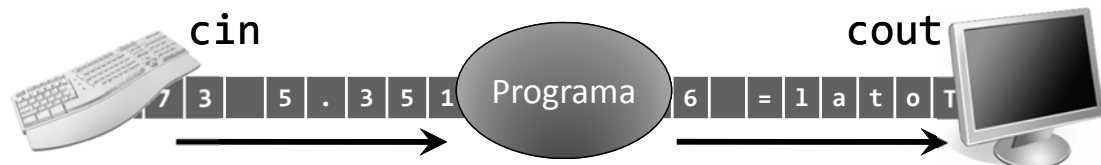
```
#include <iostream>  
using namespace std;
```

Conectan la ejecución del programa con los dispositivos de E/S

Son secuencias de caracteres

Entrada por teclado: flujo de entrada cin (tipo istream)

Salida por pantalla: flujo de salida cout (tipo ostream)





Biblioteca `iostream` con espacio de nombres `std`



Salta los *espacios en blanco* (espacios, tabuladores o saltos de línea)

- `char`  
Se lee un carácter en la variable
- `int`  
Se leen dígitos y se transforman en el valor a asignar
- `float/double`:  
Se leen dígitos (quizá el punto y más dígitos) y se asigna el valor
- `bool`:  
Si se lee `1`, se asigna `true`; con cualquier otro valor se asigna `false`

 Se amigable con el usuario   
Lee cada dato en una línea

```
cout << "Introduce tu edad: ";  
cin >> edad;
```



## Lectura de cadenas (`string`)

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

`cin >> cadena` termina con el primer espacio en blanco  
`cin.sync()` descarta la entrada pendiente

```
string nombre, apellidos;  
cout << "Nombre: ";  
cin >> nombre;  
cout << "Apellidos: ";  
cin >> apellidos;  
cout << "Nombre completo: "  
    << nombre << " "  
    << apellidos << endl;
```

```
Nombre: Luis Antonio  
Apellidos: Nombre completo: Luis Antonio
```

apellidos recibe "Antonio"

```
string nombre, apellidos;  
cout << "Nombre: ";  
cin >> nombre;  
cin.sync(); ←  
cout << "Apellidos: ";  
cin >> apellidos;  
cout << ...
```

```
Nombre: Luis Antonio  
Apellidos: Hernández Yáñez  
Nombre completo: Luis Hernández
```

*¿Cómo leer varias palabras?*

*Siguiente página...*



# Entrada por teclado

## *Lectura sin saltar los espacios en blanco iniciales*

Llamada a funciones con el operador punto (.) :

El operador punto permite llamar a una función sobre una variable *variable.función(argumentos)*

Lectura de un carácter sin saltar espacios en blanco:

```
cin.get(c); // Lee el siguiente carácter
```

Lectura de cadenas sin saltar los espacios en blanco:

```
getline(cin, cad);
```

Lee todo lo que haya hasta el final de la línea (Intro)

Recuerda:

*Espacios en blanco* son espacios, tabuladores, saltos de línea, ...



# Salida por pantalla



## *Representación textual de los datos*

```
int meses = 7;
```

```
cout << "Total: " << 123.45 << endl << " Meses: " << meses;
```

El valor `double` 123.45 se guarda en memoria en binario

Su representación textual es: '1' '2' '3' '.' '4' '5'

```
double d = 123.45;
```

d    123.45    ¡Un número real!

```
cout << d;
```

La biblioteca `iostream` define la constante `endl` como un salto de línea

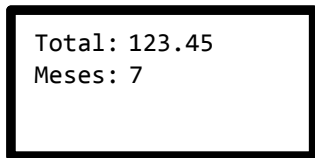
¡Un texto!  
(secuencia de caracteres)







```
int meses = 7;
cout << "Total: " << 123.45 << endl << " Meses: " << meses;
cout << 123.45 << endl << " Meses: " << meses;
cout << endl << " Meses: " << meses;
cout << " Meses: " << meses;
cout << meses;
```



Luis Hernández Yáñez



## Formato de la salida

#include <iomanip>

Constantes y funciones a enviar a cout para ajustar el formato de salida

Biblioteca	Constante/función	Propósito
iostream	showpoint / noshowpoint	Mostrar o no el punto decimal para reales sin decimales (34.0)
	fixed	Notación de punto fijo (reales) (123.5)
	scientific	Notación científica (reales) (1.235E+2)
	boolalpha	Valores bool como true / false
	left / right	Ajustar a la izquierda/derecha (por defecto)
iomanip	setw(anchura)*	Nº de caracteres (anchura) para el dato
	setprecision(p)	Precisión: Nº de dígitos (en total) Con fixed o scientific, nº de decimales

\*setw() sólo afecta al siguiente dato que se escriba, mientras que los otros afectan a todos

Luis Hernández Yáñez



# Formato de la salida

```
bool fin = false;
cout << fin << "->" << boolalpha << fin << endl;
double d = 123.45;
char c = 'x';
int i = 62;
cout << d << c << i << endl;
cout << "|" << setw(8) << d << "|" << endl;
cout << "|" << left << setw(8) << d << "|" << endl;
cout << "|" << setw(4) << c << "|" << endl;
cout << "|" << right << setw(5) << i << "|" << endl;
double e = 96;
cout << e << " - " << showpoint << e << endl;
cout << scientific << d << endl;
cout << fixed << setprecision(8) << d << endl;
```

```
0->>false

123.45x62
| 123.45|
|123.45 |
|x |
| 62|

96 - 96.0000
1.234500e+002
123.45000000
```



# Fundamentos de la programación

## Funciones definidas por el programador



# Funciones en C++

Los programas pueden incluir otras funciones además de `main()`

Forma general de una función en C++:

```
tipo nombre(parámetros) // Cabecera
{
    // Cuerpo
}
```

- ✓ *Tipo* de dato que devuelve la función como resultado
  - ✓ *Parámetros* para proporcionar datos a la función
- Declaraciones de variables separadas por comas
- ✓ *Cuerpo*: secuencia de declaraciones e instrucciones  
¡Un bloque de código!



## Datos en las funciones

- ✓ Datos locales: declarados en el cuerpo de la función  
Datos auxiliares que utiliza la función (puede no haber)
  - ✓ Parámetros: declarados en la cabecera de la función  
Datos de entrada de la función (puede no haber)
- Ambos son de uso exclusivo de la función y no se conocen fuera

```
double f(int x, int y) {
    // Declaración de datos locales:
    double resultado;

    // Instrucciones:
    resultado = 2 * pow(x, 5) + sqrt(pow(x, 3)
    / pow(y, 2)) / abs(x * y) - cos(y);

    return resultado; // Devolución del resultado
}
```

$$f(x, y) = 2x^5 + \frac{\sqrt{y^2}}{|x \times y|} - \cos(y)$$



# Argumentos

## Llamada a una función con parámetros

Nombre(Argumentos)

Al llamar a la función:

- Tantos argumentos entre los paréntesis como parámetros
- Orden de declaración de los parámetros
- Cada argumento: mismo tipo que su parámetro
- Cada argumento: expresión válida (se pasa el resultado)

Se copian los valores resultantes de las expresiones en los correspondientes parámetros

Llamadas a la función: en expresiones de otras funciones

```
int valor = f(2, 3);
```



# Paso de argumentos

## Se copian los argumentos en los parámetros

```
int funcion(int x, double a) {  
    ...  
}  
  
int main() {  
    int i = 124;  
    double d = 3;  
    funcion(i, 33 * d);  
    ...  
    return 0; // main() devuelve 0 al S.O.  
}
```

Memoria	
i	124
d	3.0
...	...
x	124
a	99.0
...	...



Los argumentos no se modifican

# Resultado de la función

## *La función ha de devolver un resultado*

La función termina su ejecución devolviendo un resultado

La instrucción `return` (*sólo una en cada función*)

- Devuelve el dato que se pone a continuación (tipo de la función)
- Termina la ejecución de la función

El dato devuelto sustituye a la llamada de la función:

```
int cuad(int x) {  
    return x * x;  
    x = x * x;  
}  
  
int main() {  
    cout << 2 * cuad(16);  
    return 0; 256  
}
```

Esta instrucción no se ejecutará nunca



# Prototipos de las funciones

## *¿Qué funciones hay en el programa?*

Colocaremos las funciones después de `main()`

*¿Son correctas las llamadas a funciones del programa?*

- ¿Existe la función?
- ¿Concuerdan los argumentos con los parámetros?

→ Prototipos tras las inclusiones de bibliotecas

Prototipo de función: Cabecera de la función terminada en ;

```
double f(int x, int y);  
int funcion(int x, double a)  
int cuad(int x);  
...
```

`main()` es la única función que no hay que prototipar



# Un programa con funciones

---

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

// Prototipos de las funciones (excepto main())
bool par(int num);
bool letra(char car);
int suma(int num);
double formula(int x, int y);

int main() {
    int numero, sum, x, y;
    char caracter;
    double f;
    cout << "Entero: ";
    cin >> numero;
    if (par(numero)) {
        cout << "Par";
    }
    ...
}
```



# Un programa con funciones

---

```
else {
    cout << "Impar";
}
cout << endl;
if (numero > 1) {
    cout << "Sumatorio de 1 a " << numero << ": "
        << suma(numero) << endl;
}
cout << "Carácter: ";
cin >> caracter;
if (!letra(caracter)) {
    cout << "no ";
}
cout << "es una letra" << endl;
cout << "f(x, y) = " << formula(x, y) << endl;
// Los argumentos pueden llamarse igual o no que los parámetros

return 0;
}
...
```



# Un programa con funciones

---

```
// Implementación de las funciones propias
```

```
bool par(int num) {  
    bool esPar;  
  
    if (num % 2 == 0) {  
        esPar = true;  
    }  
    else {  
        esPar = false;  
    }  
  
    return esPar;  
}  
...
```

Luis Hernández Yáñez



# Un programa con funciones

---

```
bool letra(char car) {  
    bool esLetra;  
    if ((car >= 'a') && (car <= 'z') || (car >= 'A') && (car <= 'Z')) {  
        esLetra = true;  
    }  
    else {  
        esLetra = false;  
    }  
    return esLetra;  
}  
  
int suma(int num) {  
    int sum = 0, i = 1;  
    while (i < num) {  
        sum = sum + i;  
        i++;  
    }  
    return sum;  
}  
...
```

Luis Hernández Yáñez



```
double formula(int x, int y) {  
    double f;  
  
    f = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))  
        / abs(x * y) - cos(y);  
  
    return f;  
}
```






## Acerca de *Creative Commons*



### *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.







ANEXO

## Tipos: Detalles técnicos

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

<code>int</code>	214
<code>float</code>	216
Notación científica	217
<code>double</code>	218
<code>char</code>	220
<code>bool</code>	221
<code>string</code>	222
Literales con especificación de tipo	223

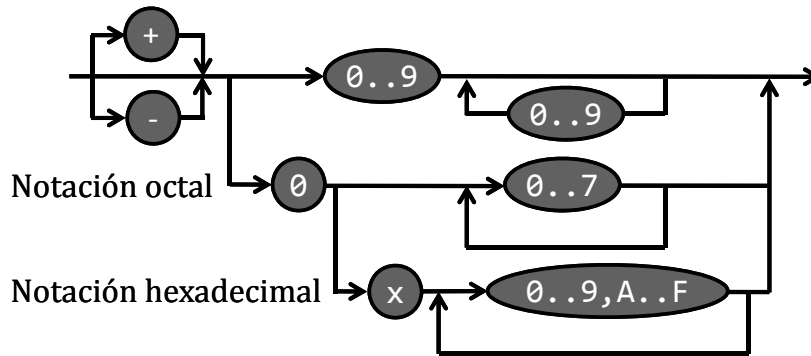
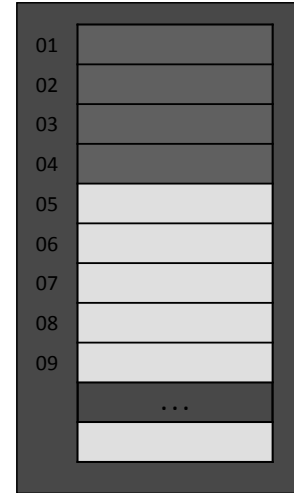


Intervalo de valores:

-2147483648 .. 2147483647

Bytes de memoria: 4\* (\*Depende de la máquina  
4 bytes es lo más habitual)

Literales:  
1363, -12, 010, 0x1A  
Se puede saber cuántos se usan con la función  
sizeof(int)



Luis Hernández Yáñez



Números en notación octal (base 8: dígitos entre 0 y 7):

-010 = -8 en notación decimal

$$10 = 1 \times 8^1 + 0 \times 8^0 = 1 \times 8 + 0$$

0423 = 275 en notación decimal

$$423 = 4 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 = 4 \times 64 + 2 \times 8 + 3 = 256 + 16 + 3$$

Números en notación hexadecimal (base 16):

Dígitos posibles: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

0x1F = 31 en notación decimal

$$1F = 1 \times 16^1 + F \times 16^0 = 1 \times 16 + 15$$

0xAD = 173 en notación decimal

$$AD = A \times 16^1 + D \times 16^0 = 10 \times 16 + 13 = 160 + 13$$

Luis Hernández Yáñez



Intervalo de valores:

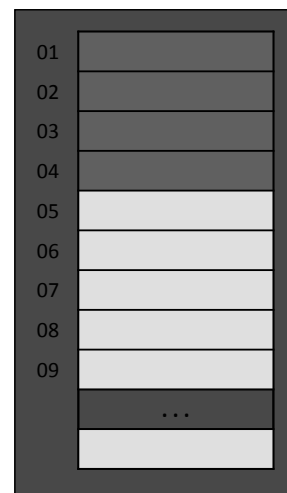
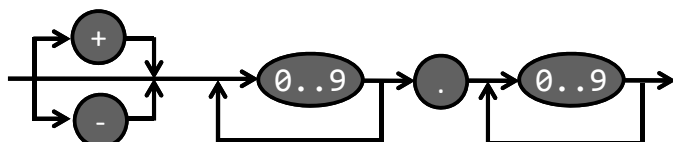
+/- 1.18e-38 .. 3.40e+38

Bytes de memoria: 4\* (\* sizeof(float))

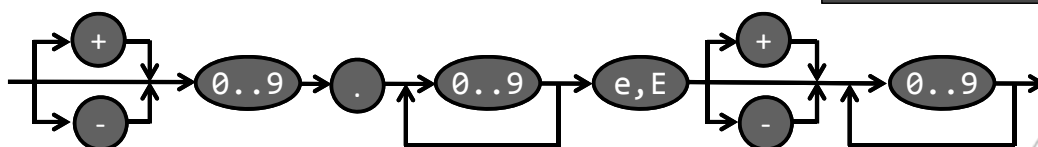
Punto flotante. Precisión: 7 dígitos

Literales (punto decimal):

✓ Notación normal: 134.45, -1.1764



✓ Notación científica: 1.4E2, -5.23e-02



## Notación científica

Siempre un número (con o sin signo) con un solo dígito de parte entera, seguido del exponente (potencia de 10):

-5.23e-2 → -5,23 x 10<sup>-2</sup> → -0,0523

1.11e2 → 1,11 x 10<sup>2</sup> → 111,0

7.4523e-04 → 7,4523 x 10<sup>-4</sup> → 0,00074523

-3.3333e+06 → -3,3333 x 10<sup>6</sup> → -3.333.300



# double

## Números reales (con decimales)

Intervalo de valores:

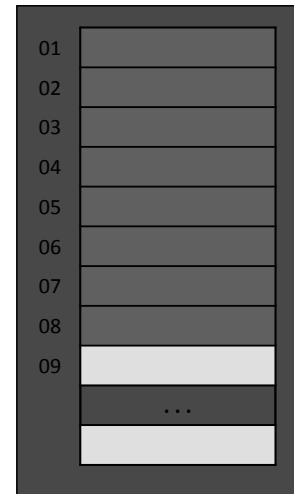
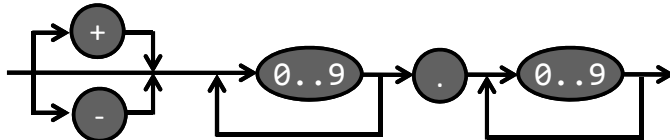
+/- 2.23e-308 .. 1.79e+308

Bytes de memoria: 8\* (\* sizeof(double))

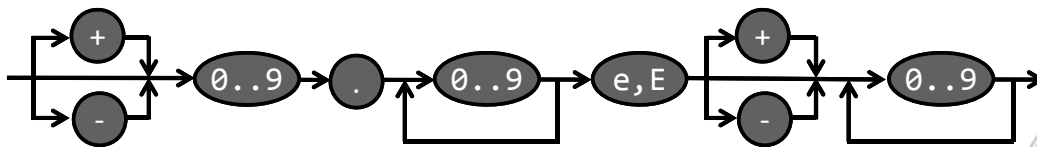
Punto flotante. Precisión: 15 dígitos

Literales (punto decimal):

✓ Notación normal: 134.45, -1.1764



✓ Notación científica: 1.4E2, -5.23e-02



Luis Hernández Yáñez



# char

## Caracteres

Intervalo de valores:

Juego de caracteres (ASCII)

Bytes de memoria: 1 (FC)

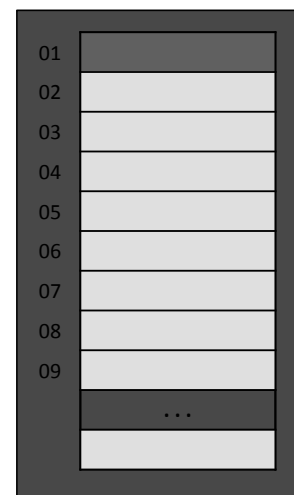
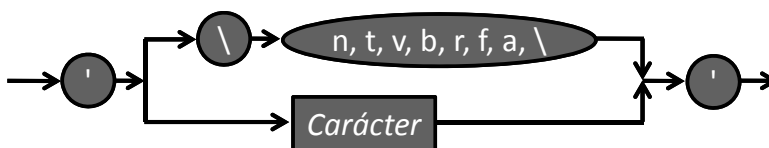
Literales:

'a', '%', '\t'

Constantes de barra invertida:  
(0 secuencias de escape)

Para caracteres de control

'\t' = tabulador, '\n' = salto de línea, ...



Luis Hernández Yáñez



# char

Juego de caracteres ASCII:

*American Standard Code for Information Interchange* (1963)

Caracteres con códigos entre 0 y 127 (7 bits)

- Caracteres de control:  
Códigos del 0 al 31 y 127  
Tabulación, salto de línea,...
- Caracteres imprimibles:  
Códigos del 32 al 126

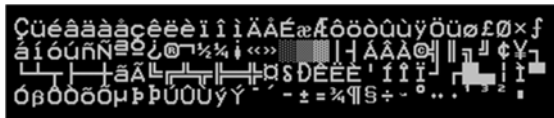
```
!"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

Juego de caracteres ASCII extendido (8 bits):

ISO-8859-1

+ Códigos entre 128 y 255

Multitud de codificaciones:  
EBCDIC, UNICODE, UTF-8, ...



# bool

*Valores lógicos*

Sólo dos valores posibles:

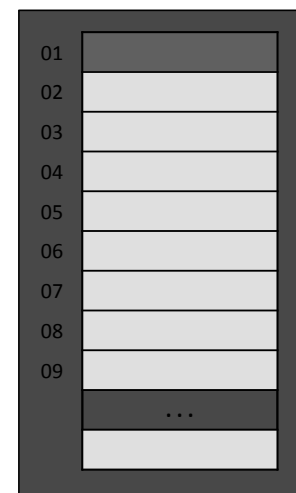
- Verdadero (*true*)
- Falso (*false*)

Bytes de memoria: 1 (FC)

Literales:

`true`, `false`

En realidad, cualquier número distinto de 0 es equivalente a `true` y el número 0 es equivalente a `false`



"Hola", "Introduce el numerador: ", "X142FG5TX?%A"



Secuencias de caracteres

Se asigna la memoria que se necesita para la secuencia concreta

Requieren la biblioteca string con el espacio de nombres std:

```
#include <string>
```

```
using namespace std;
```



*¡Ojo!*

Las comillas tipográficas (apertura/cierre) “...” te darán problemas al compilar. Asegúrate de utilizar comillas rectas: “...”



## Literales con especificación de tipo

Por defecto un literal entero se considera un dato `int`

– `long int`: 35L, 1546l

– `unsigned int`: 35U, 1546u

– `unsigned long int`: 35UL, 1546ul

Por defecto un literal real se considera un dato `double`

– `float`: 1.35F, 15.46f

– `long double`: 1.35L, 15.46l

### *Abreviaturas para modificadores de tipos*

`short`  $\equiv$  `short int`

`long`  $\equiv$  `long int`

Es preferible evitar el uso de tales abreviaturas:

*Minimizar la cantidad de información a recordar sobre el lenguaje*








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.



## 3

## Tipos e instrucciones II

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Tipos, valores y variables	227	El bucle for	321
Conversión de tipos	232	Bucles anidados	331
Tipos declarados por el usuario	236	Ámbito y visibilidad	339
Tipos enumerados	238	Secuencias	349
Entrada/Salida		Recorrido de secuencias	355
con archivos de texto	248	Secuencias calculadas	363
Lectura de archivos de texto	253	Búsqueda en secuencias	370
Escritura en archivos de texto	266	Arrays de datos simples	374
Flujo de ejecución	272	Uso de variables arrays	379
Selección simple	276	Recorrido de arrays	382
Operadores lógicos	282	Búsqueda en arrays	387
Anidamiento de if	286	Arrays no completos	393
Condiciones	290		
Selección múltiple	293		
La escala if-else-if	295		
La instrucción switch	302		
Repetición	313		
El bucle while	316		





## Tipos, valores y variables



## Tipos, valores y variables

---

### *Tipo*

Conjunto de valores con sus posibles operaciones

### *Valor*

Conjunto de bits interpretados como de un tipo concreto

### *Variable (o constante)*

Cierta memoria con nombre para valores de un tipo

### *Declaración*

Instrucción que identifica un nombre

### *Definición*

Declaración que asigna memoria a una variable o constante



# Variables

---

Memoria suficiente para su tipo de valores

short int i = 3;

i 3

int j = 9;

j 9

char c = 'a';

c a

double x = 1.5;

x 1.5

El significado de los bits depende del tipo de la variable:

00000000 00000000 00000000 01111000

Interpretado como int es el entero 120

Interpretado como char (sólo 01111000) es el carácter 'x'



# Tipos

---

## ✓ Simples

❖ Estándar: int, float, double, char, bool  
Conjunto de valores predeterminado

❖ Definidos por el usuario: *enumerados*  
Conjunto de valores definido por el programador

## ✓ Estructurados (Tema 5)

❖ Colecciones homogéneas: *arrays*  
Todos los elementos de la colección de un mismo tipo

❖ Colecciones heterogéneas: *estructuras*  
Elementos de la colección de tipos distintos



# Tipos simples estándar

---

Con sus posibles modificadores:

`[unsigned] [short] int`

`long long int`

`long int ≡ int`

`float`

`[long] double`

`char`

`bool`

Definición de variables:

*tipo nombre [ = expresión] [, ...];*

Definición de constantes con nombre:

*const tipo nombre = expresión;*



# Fundamentos de la programación

---

## Conversión de tipos



# Conversiones automáticas de tipos

## Promoción de tipos

Dos operandos de tipos distintos:

El valor del tipo *menor* se promociona al tipo *mayor*

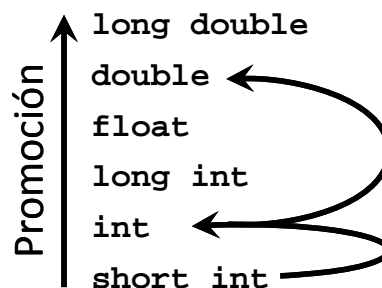
```
short int i = 3;  
int j = 2;  
double a = 1.5, b;  
b = a + i * j;
```

```
b = a + 3 * 2;
```

↳ Valor 3 short int (2 bytes) → int (4 bytes)

```
b = 1.5 + 6;
```

↳ Valor 6 int (4 bytes) → double (8 bytes)



# Conversiones seguras y no seguras

Conversión segura:

De un tipo menor a un tipo mayor

short int → int → long int → ...

Conversión no segura:

De un tipo mayor a un tipo menor

```
int entero = 1234;
```

```
char caracter;
```

```
caracter = entero; // Conversión no segura
```

Menor memoria: Pérdida de información en la conversión

long double

double

float

long int

int

short int



# Moldes (*casts*)

---

Fuerzan una conversión de tipo:

*tipo*(*expresión*)

El valor resultante de la *expresión* se trata como un valor del *tipo*

```
int a = 3, b = 2;  
cout << a / b;           // Muestra 1 (división entera)  
cout << double(a) / b; // Muestra 1.5 (división real)
```

Tienen la mayor prioridad



# Fundamentos de la programación

---

## Tipos declarados por el usuario



# Tipos declarados por el usuario

---

Describimos los valores de las variables del tipo  
`typedef descripción nombre_de_tipo;`

↑  
*Identificador válido*



Nombres de tipos propios:

t minúscula seguida de una o varias palabras capitalizadas

Los colorearemos en naranja, para remarcar que son tipos

```
typedef descripción tMiTipo;  
typedef descripción tMoneda;  
typedef descripción tTiposDeCalificacion;
```

*Declaración de tipo frente a definición de variable*



# Fundamentos de la programación

---

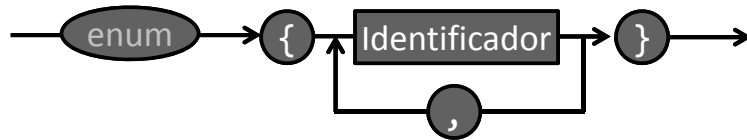
## Tipos enumerados



# Enumeraciones

Enumeración del conjunto de valores posibles para las variables:

```
enum { símbolo1, símbolo2, ..., símboloN }
```



```
enum { centimo, dos_centimos, cinco_centimos,  
      diez_centimos, veinte_centimos,  
      medio_euro, euro }
```

Valores literales que pueden tomar las variables (en amarillo)



## Tipos enumerados

Mejoran la legibilidad

```
typedef descripción nombre_de_tipo;
```

Elegimos un nombre para el tipo: tMoneda

*descripción*

```
typedef enum { centimo, dos_centimos, cinco_centimos,  
             diez_centimos, veinte_centimos,  
             medio_euro, euro } tMoneda;
```

En el ámbito de la declaración, se reconoce un nuevo tipo tMoneda

```
tMoneda moneda1, moneda2;
```

Cada variable de ese tipo contendrá alguno de los símbolos

```
moneda1 = dos_centimos;
```

```
moneda2 = euro;
```

moneda1 dos\_centimos

moneda2 euro

(Internamente se usan enteros)



# Entrada/salida para tipos enumerados

```
typedef enum { enero, febrero, marzo, abril, mayo,  
             junio, julio, agosto, septiembre, octubre,  
             noviembre, diciembre } tMes;
```

```
tMes mes;
```

Lectura de la variable mes:

```
cin >> mes;
```

Se espera un valor entero

No se puede escribir directamente enero o junio

Y si se escribe la variable en la pantalla:

```
cout << mes;
```

Se verá un número entero

→ Código de entrada/salida específico



# Lectura del valor de un tipo enumerado

```
typedef enum { enero, febrero, marzo, abril, mayo, junio, julio,  
             agosto, septiembre, octubre, noviembre, diciembre } tMes;
```

```
int op;  
cout << " 1 - Enero" << endl;  
cout << " 2 - Febrero" << endl;  
cout << " 3 - Marzo" << endl;  
cout << " 4 - Abril" << endl;  
cout << " 5 - Mayo" << endl;  
cout << " 6 - Junio" << endl;  
cout << " 7 - Julio" << endl;  
cout << " 8 - Agosto" << endl;  
cout << " 9 - Septiembre" << endl;  
cout << "10 - Octubre" << endl;  
cout << "11 - Noviembre" << endl;  
cout << "12 - Diciembre" << endl;  
cout << "Numero de mes: ";  
cin >> op;  
tMes mes = tMes(op - 1);
```





# Escritura de variables de tipos enumerados

```
typedef enum { enero, febrero, marzo, abril, mayo, junio, julio, agosto, septiembre, octubre, noviembre, diciembre } tMes;
```

```
if (mes == enero) {
    cout << "enero";
}
if (mes == febrero) {
    cout << "febrero";
}
if (mes == marzo) {
    cout << "marzo";
}
...
if (mes == diciembre) {
    cout << "diciembre";
}
```

También podemos utilizar una instrucción switch



## Tipos enumerados

Conjunto de valores ordenado (posición en la enumeración)

```
typedef enum { lunes, martes, miercoles, jueves, viernes, sabado, domingo } tDiaSemana;
```

```
tDiaSemana dia;
```

```
...
```

```
if (dia == jueves)...
```

```
bool noLaborable = (dia >= sabado);
```

```
lunes < martes < miercoles < jueves
viernes < sabado < domingo
```

No admiten operadores de incremento y decremento

Emulación con moldes:

```
int i = int(dia); // ¡dia no ha de valer domingo!
```

```
i++;
```

```
dia = tDiaSemana(i);
```



# Ejemplo de tipos enumerados

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



Si los tipos se usan en varias funciones, los declaramos antes de los prototipos

```
typedef enum { enero, febrero, marzo, abril, mayo,
             junio, julio, agosto, septiembre, octubre,
             noviembre, diciembre } tMes;
```

```
typedef enum { lunes, martes, miercoles, jueves,
             viernes, sabado, domingo } tDiaSemana;
```

```
string cadMes(tMes mes);
string cadDia(tDiaSemana dia);
```

```
int main() {
    tDiaSemana hoy = lunes;
    int dia = 21;
    tMes mes = octubre;
    int anio = 2013;
    ...
}
```



# Ejemplo de tipos enumerados

```
// Mostramos la fecha
cout << "Hoy es: " << cadDia(hoy) << " " << dia
     << " de " << cadMes(mes) << " de " << anio
     << endl;
```

```
cout << "Pasada la medianoche..." << endl;
dia++;
int i = int(hoy);
i++;
hoy = tDiaSemana(i);
```

```
// Mostramos la fecha
cout << "Hoy es: " << cadDia(hoy) << " " << dia
     << " de " << cadMes(mes) << " de " << anio
     << endl;
```

```
return 0;
}
```



```
string cadMes(tMes mes) {
    string cad;

    if (mes == enero) {
        cad = "enero";
    }
    if (mes == febrero) {
        cad = "febrero";
    }
    ...
    if (mes == diciembre) {
        cad = "diciembre";
    }

    return cad;
}

string cadDia(tDiaSemana dia);
string cad;

if (dia == lunes) {
    cad = "lunes";
}
if (dia == martes) {
    cad = "martes";
}
...
if (dia == domingo) {
    cad = "domingo";
}

return cad;
}
```



## Fundamentos de la programación

# Entrada/Salida con archivos de texto



# Archivos

Datos del programa: en la memoria principal (volátil)

Medios (dispositivos) de almacenamiento permanente:

- Discos magnéticos fijos (internos) o portátiles (externos)
- Cintas magnéticas
- Discos ópticos (CD, DVD, BlueRay)
- Memorias USB
- ...



Mantienen la información en archivos  
Secuencias de datos



## Archivos de texto y archivos binarios

Archivo de texto: secuencia de caracteres

T	o	t	a	l	:		1	2	3	.	4	↵	A	...
---	---	---	---	---	---	--	---	---	---	---	---	---	---	-----

Archivo binario: contiene una secuencia de códigos binarios

A0	25	2F	04	D6	FF	00	27	6C	CA	49	07	5F	A4	...
----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----

(Códigos representados en notación hexadecimal)

Los archivos se manejan en los programas por medio de *flujos*

Archivos de texto: *flujos de texto*

Similar a la E/S por consola

(Más adelante veremos el uso de archivos binarios)



# Archivos de texto

---

Textos dispuestos en sucesivas líneas

Carácter de fin de línea entre línea y línea (Intro)

Posiblemente varios datos en cada línea

Ejemplo: Compras de los clientes

En cada línea, NIF del cliente, unidades compradas, precio unitario y descripción de producto, separados por espacio

12345678F 2 123.95 Reproductor de DVD↓

00112233A 1 218.4 Disco portátil↓

32143567J 3 32 Memoria USB 16Gb↓

76329845H 1 134.5 Modem ADSL↓

...

Normalmente terminan con un dato especial (*centinela*)

Por ejemplo, un NIF que sea X



## Flujos de texto para archivos

---

`#include <fstream>`

- ✓ Lectura del archivo: flujo de entrada
- ✓ Escritura en el archivo: flujo de salida

No podemos leer y escribir en un mismo flujo

Un flujo de texto se puede utilizar para lectura o para escritura:

- Flujos (archivos) de entrada: variables de tipo `ifstream`
- Flujos (archivos) de salida : variables de tipo `ofstream`

Biblioteca `fstream` (sin espacio de nombres)



## Lectura de archivos de texto



## Lectura de archivos de texto

---

*Flujos de texto de entrada*

`ifstream`

Para leer de un archivo de texto:

- 1 Declara una variable de tipo `ifstream`
- 2 Asocia la variable con el archivo de texto (*apertura del archivo*)
- 3 Realiza las operaciones de lectura
- 4 Desliga la variable del archivo de texto (*cierre el archivo*)



# Lectura de archivos de texto

---

## Apertura del archivo

Conecta la variable con el archivo de texto del dispositivo

```
flujo.open(cadena_literal);  
ifstream archivo;  
archivo.open("abc.txt");  
if (archivo.is_open()) ...
```

*¡El archivo debe existir!*

```
is_open():  
true si el archivo  
se ha podido abrir  
false en caso contrario
```

## Cierre del archivo

Desconecta la variable del archivo de texto del dispositivo

```
flujo.close();  
archivo.close();
```



# Lectura de archivos de texto

---

## Operaciones de lectura

✓ Extractor (>>) `archivo >> variable;`

Salta primero los espacios en blanco (espacio, tab, Intro, ...)

Datos numéricos: lee hasta el primer carácter no válido

Cadenas (`string`): lee hasta el siguiente espacio en blanco

✓ `archivo.get(c)`

Lee el siguiente carácter en la variable `c`, sea el que sea

✓ `getline(archivo, cadena)`

Lee en la `cadena` todos los caracteres que queden en la línea

Incluidos los espacios en blanco

Hasta el siguiente salto de línea (descartándolo)

Con los archivos no tiene efecto la función `sync()`



# Lectura de archivos de texto

---

## *¿Qué debo leer?*

- ✓ Un número  
Usa el extractor `archivo >> num;`
- ✓ Un carácter (sea el que sea)  
Usa la función `get()` `archivo.get(c);`
- ✓ Una cadena sin espacios  
Usa el extractor `archivo >> cad;`
- ✓ Una cadena posiblemente con espacios  
Usa la función `getline()` `getline(archivo, cad);`



# Lectura de archivos de texto

---

## *¿Dónde queda pendiente la entrada?*

- ✓ Número leído con el extractor  
En el primer carácter no válido (inc. espacios en blanco)
- ✓ Carácter leído con `get()`  
En el siguiente carácter (inc. espacios en blanco)
- ✓ Una cadena leída con el extractor  
En el siguiente espacio en blanco
- ✓ Una cadena leída con la función `getline()`  
Al principio de la siguiente línea

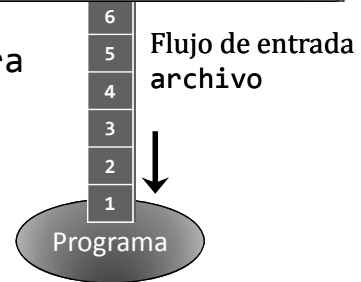




# Lectura de archivos de texto

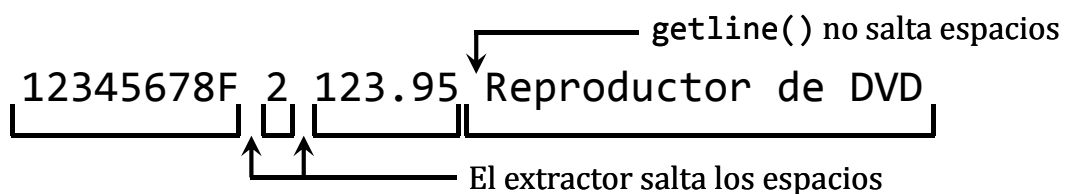
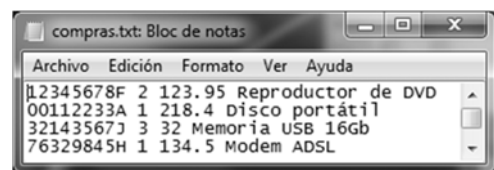
```
string nif, producto;  
int unidades;  
double precio;  
char aux;
```

- 1 ifstream archivo;
- 2 archivo.open("compras.txt"); // Apertura
- 3 archivo >> nif >> unidades >> precio;  
getline(archivo, producto);
- 4 archivo.close(); // Cierre



# Lectura de archivos de texto

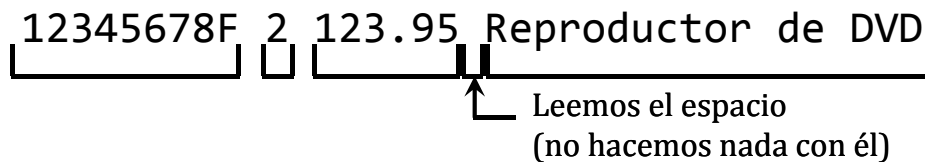
```
archivo >> nif;  
archivo >> unidades;  
archivo >> precio;  
getline(archivo, producto);
```



# Lectura de archivos de texto

```
archivo >> nif;  
archivo >> unidades;  
archivo >> precio;  
archivo.get(aux); // Salta el espacio en blanco  
getline(archivo, producto);
```

12345678F 2 123.95 Reproductor de DVD



Leemos el espacio  
(no hacemos nada con él)

nif 12345678F unidades 2  
producto Reproductor de DVD precio 123.95

Sin espacio

Luis Hernández Yáñez



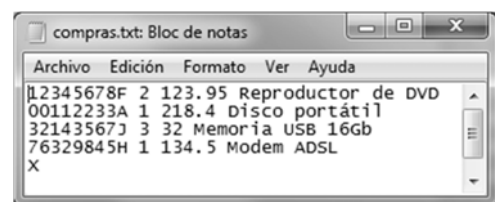
Fundamentos de la programación: Tipos e instrucciones II

Página 261



# Procesamiento de los datos de un archivo

Cada línea, datos de una compra  
Mostrar el total de cada compra  
unidades x precio más IVA (21%)  
Final: "X" como NIF



Bucle de procesamiento:

- ✓ Cada paso del bucle (ciclo) procesa una línea (compra)
- ✓ Podemos usar las mismas variables en cada ciclo

*Leer primer NIF*

*Mientras el NIF no sea X:*

*Leer unidades, precio y descripción*

*Calcular y mostrar el total*

*Leer el siguiente NIF*

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

Página 262



# Procesamiento de los datos de un archivo

leer.cpp

```
#include <iostream>
#include <string>
using namespace std;
#include <fstream>
#include <iomanip> // Formato de salida

int main() {
    const int IVA = 21;
    string nif, producto;
    int unidades;
    double precio, neto, total, iva;
    char aux;
    ifstream archivo;
    int contador = 0;

    archivo.open("compras.txt"); // Apertura
    ...
```

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

Página 263



# Procesamiento de los datos de un archivo

```
if (archivo.is_open()) { // Existe el archivo
    archivo >> nif; // Primer NIF
    while (nif != "X") {
        archivo >> unidades >> precio;
        archivo.get(aux); // Salta el espacio
        getline(archivo, producto);
        contador++;
        neto = unidades * precio;
        iva = neto * IVA / 100;
        total = neto + iva;
        cout << "Compra " << contador << ".-" << endl;
        cout << "    " << producto << ": " << unidades
            << " x " << fixed << setprecision(2)
            << precio << " = " << neto << " - I.V.A.: "
            << iva << " - Total: " << total << endl;
        archivo >> nif; // Siguiendo NIF
    } ...
```

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

Página 264



# Procesamiento de los datos de un archivo

---

```
        archivo.close(); // Cierre
    }
    else {
        cout << "ERROR: No se ha podido abrir el archivo"
            << endl;
    }
    return 0;
}
```

```
Compra 1.-
Reproductor de DVD: 2 x 123.95 = 247.90 - I.V.A.: 52.06 - Total: 299.96
Compra 2.-
Disco portatil: 1 x 218.40 = 218.40 - I.V.A.: 45.86 - Total: 264.26
Compra 3.-
Memoria USB 16Gb: 3 x 32.00 = 96.00 - I.V.A.: 20.16 - Total: 116.16
Compra 4.-
Modem ADSL: 1 x 134.50 = 134.50 - I.V.A.: 28.25 - Total: 162.75
```



# Fundamentos de la programación

---

## Escritura en archivos de texto



# Escritura en archivos de texto

## Flujos de texto de salida

ofstream

Para crear un archivo de texto y escribir en él:

- 1 Declara una variable de tipo ofstream
- 2 Asocia la variable con el archivo de texto (*crea el archivo*)
- 3 Realiza las escrituras por medio del operador << (insertor)
- 4 Desliga la variable del archivo de texto (*cierra el archivo*)



**¡Atención!**

Si el archivo ya existe, se borra todo lo que hubiera



**¡Atención!**

Si no se cierra el archivo se puede perder información



# Escritura en archivos de texto

```
int valor = 999;
```

- 1 ofstream archivo;
- 2 archivo.open("output.txt"); // Apertura
- 3 archivo << 'X' << " Hola! " << 123.45  
    << endl << valor << "Bye!";
- 4 archivo.close(); // Cierre

Programa

2  
1  
!  
a  
l  
o  
H  
X

Flujo de salida  
archivo



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

int main() {
    string nif, producto;
    int unidades;
    double precio;
    char aux;
    ofstream archivo;

    archivo.open("output.txt"); // Apertura (creación)

    cout << "NIF del cliente (X para terminar): ";
    cin >> nif;
    ...
}
```



## Escritura en archivos de texto

```
while (nif != "X") {
    // Queda pendiente el Intro anterior...
    cin.get(aux); // Leemos el Intro
    cout << "Producto: ";
    getline(cin, producto);
    cout << "Unidades: ";
    cin >> unidades;
    cout << "Precio: ";
    cin >> precio;
    // Escribimos los datos en una línea del archivo...
    // Con un espacio de separación entre ellos
    archivo << nif << " " << unidades << " "
        << precio << " " << producto << endl;
    cout << "NIF del cliente (X para terminar): ";
    cin >> nif;
}
...
```



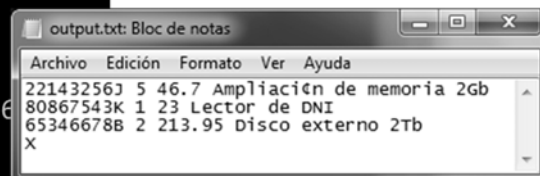
# Escritura en archivos de texto

---

```
// Escribimos el centinela final...
archivo << "X";
archivo.close(); // Cierre del archivo

return 0;
}
```

```
NIF del cliente (X para terminar): 22143256J
Producto: Ampliación de memoria 2Gb
Unidades: 5
Precio: 46.7
NIF del cliente (X para terminar): 80867543K
Producto: Lector de DNI
Unidades: 1
Precio: 23
NIF del cliente (X para terminar): 65346678B
Producto: Disco externo 2Tb
Unidades: 2
Precio: 213.95
NIF del cliente (X para terminar): X
```



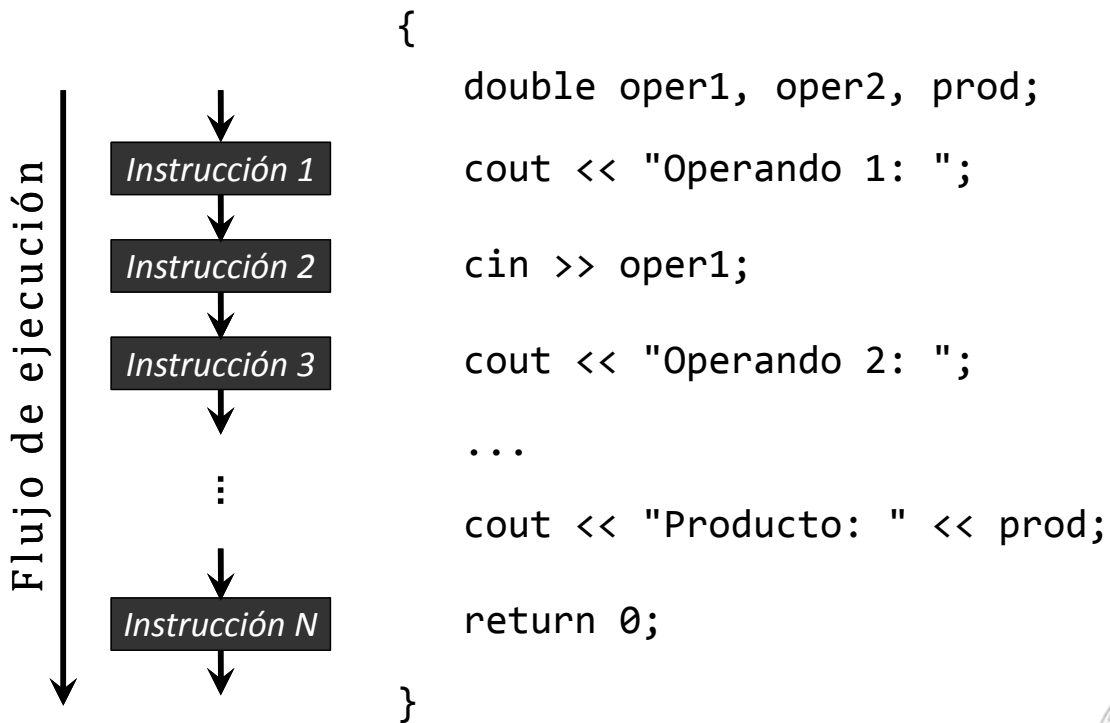
# Fundamentos de la programación

---

## Flujo de ejecución



# Ejecución secuencial



Luis Hernández Yáñez



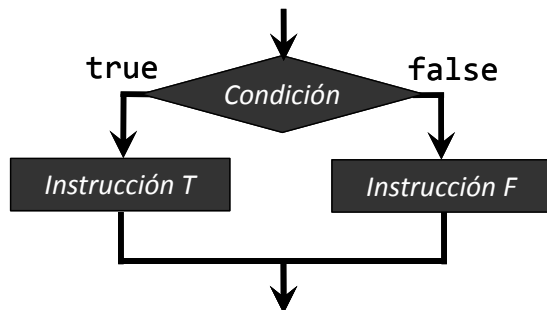
# Selección



*Uno entre dos o más caminos de ejecución*

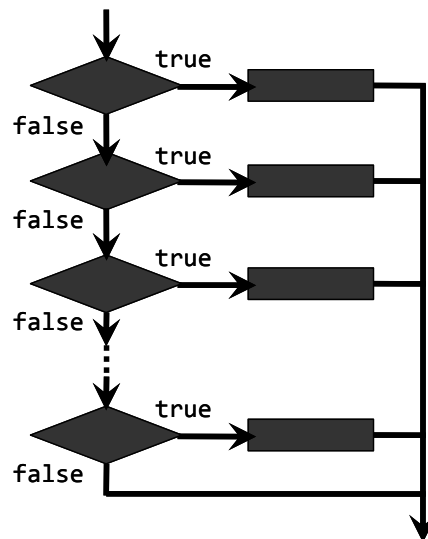
Selección simple (2 caminos)

Selección múltiple (> 2 caminos)



if

if-else-if  
switch



*Diagramas de flujo*

Luis Hernández Yáñez



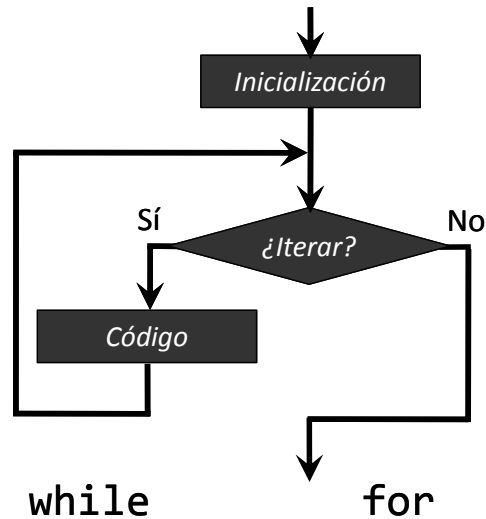


# Repetición (iteración)



*Repetir la ejecución de una o más instrucciones*

Acumular, procesar colecciones, ...



Luis Hernández Yáñez



# Fundamentos de la programación

## Selección simple

Luis Hernández Yáñez



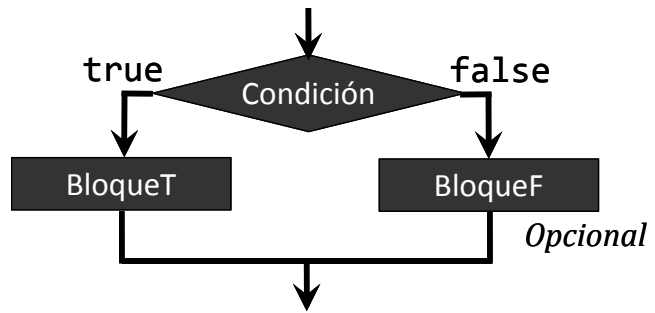
# Selección simple (bifurcación)



## La instrucción if

```
if (condición) {  
    ↪ códigoT  
}  
[else {  
    ↪ códigoF  
}]
```

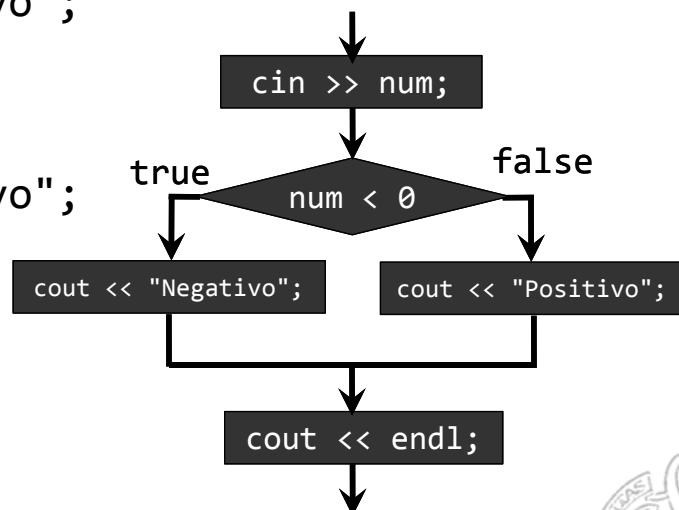
condición: expresión bool  
Cláusula else opcional



# La instrucción if

signo.cpp

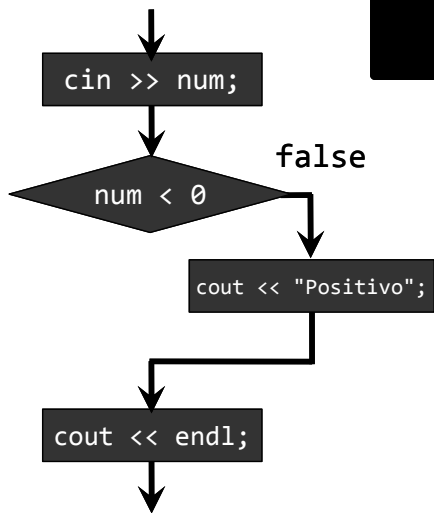
```
int num;  
cin >> num;  
if (num < 0) {  
    cout << "Negativo";  
}  
else {  
    cout << "Positivo";  
}  
cout << endl;
```



# La instrucción if

```
int num;  
cin >> num;  
if (num < 0) {  
    cout << "Negativo";  
}  
else {  
    cout << "Positivo";  
}  
cout << endl;
```

```
129  
Positivo
```



num 129

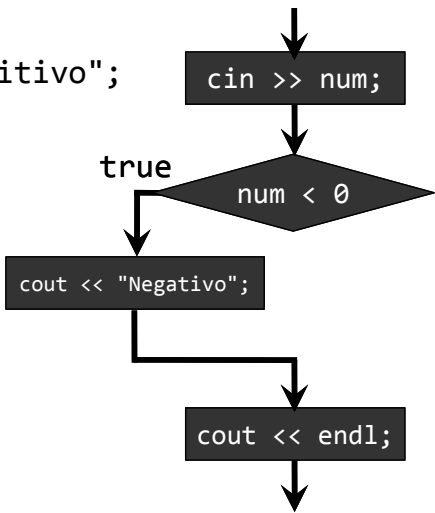
Luis Hernández Yáñez



# La instrucción if

```
int num;  
cin >> num;  
if (num < 0) {  
    cout << "Negativo";  
}  
else {  
    cout << "Positivo";  
}  
cout << endl;
```

```
-5  
Negativo  
—
```



num -5

Luis Hernández Yáñez



División entre dos números protegida frente a intento de división por 0

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

int main() {
    double numerador, denominador, resultado;
    cout << "Numerador: ";
    cin >> numerador;
    cout << "Denominador: ";
    cin >> denominador;
    if (denominador == 0) {
        cout << "Imposible dividir entre 0!";
    }
    else {
        resultado = numerador / denominador;
        cout << "Resultado: " << resultado << endl;
    }
    return 0;
}
```

Luis Hernández Yáñez



## Fundamentos de la programación

# Operadores lógicos (condiciones compuestas)

Luis Hernández Yáñez



# Operadores lógicos (booleanos)

Se aplican a valores bool (*condiciones*)

El resultado es de tipo bool

!	NO	Monario
&&	Y	Binario
	O	Binario

Operadores (prioridad)
...
!
* / %
+ -
< <= > >=
== !=
&&

Luis Hernández Yáñez



## Operadores lógicos - Tablas de verdad

		!	&&			
			true	false	true	false
true	false	false	true	false	true	true
false	true	true	false	false	false	false

NO (*Not*)

Y (*And*)

O (*Or*)

```
bool cond1, cond2, resultado;
int a = 2, b = 3, c = 4;
resultado = !(a < 5);           // !(2 < 5) → !true → false
cond1 = (a * b + c) >= 12;     // 10 >= 12 → false
cond2 = (a * (b + c)) >= 12;   // 14 >= 12 → true
resultado = cond1 && cond2;     // false && true → false
resultado = cond1 || cond2;    // false || true → true
```

Luis Hernández Yáñez



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

int main()
{
    int num;
    cout << "Introduce un número entre 1 y 10: ";
    cin >> num;
    if ((num >= 1) && (num <= 10)) {
        cout << "Número dentro del intervalo de valores válidos";
    }
    else {
        cout << "Número no válido!";
    }
    return 0;
}
```

¡Encierra las condiciones simples entre paréntesis!

### Condiciones equivalentes

- ((num >= 1) && (num <= 10))
- ((num > 0) && (num < 11))
- ((num >= 1) && (num < 11))
- ((num > 0) && (num <= 10))



# Fundamentos de la programación

## Anidamiento de if



```
int mes, anio, dias;
cout << "Número de mes: ";
cin >> mes;
cout << "Año: ";
cin >> anio;
if (mes == 2) {
    if (bisiesto(mes, anio)) {
        dias = 29;
    }
    else {
        dias = 28;
    }
}
else {
    if ((mes == 1) || (mes == 3) || (mes == 5) || (mes == 7)
        || (mes == 8) || (mes == 10) || (mes == 12)) {
        dias = 31;
    }
    else {
        dias = 30;
    }
}
}
```

Luis Hernández Yáñez



## ¿Año bisiesto?

*Calendario Gregoriano*: bisiesto si divisible por 4, excepto el último de cada siglo (divisible por 100), salvo que sea divisible por 400

```
bool bisiesto(int mes, int anio) {
    bool esBisiesto;
    if ((anio % 4) == 0) { // Divisible por 4
        if (((anio % 100) == 0) && ((anio % 400) != 0)) {
            // Pero último de siglo y no múltiplo de 400
            esBisiesto = false;
        }
        else {
            esBisiesto = true; // Año bisiesto
        }
    }
    else {
        esBisiesto = false;
    }
    return esBisiesto;
}
```

Luis Hernández Yáñez



# Asociación de cláusulas else

Cada else se asocia al if anterior más cercano sin asociar (mismo bloque)

```
if (condición1) {  
    if (condición2) {...}  
    else {...}  
}  
else {  
    if (condición3) {  
        if (condición4) {...}  
        if (condición5) {...}  
        else {...}  
    }  
    else { ...  
}
```

Una mala sangría puede confundir

```
if (x > 0) {  
    if (y > 0) {...}  
    else {...}
```



```
if (x > 0) {  
    if (y > 0) {...}  
    else {...}
```

La sangría ayuda a asociar los else con sus if



# Fundamentos de la programación

## Condiciones





# Condiciones

- Condición simple: Expresión lógica (true/false)  
Sin operadores lógicos

```
num < 0  
car == 'a'  
isalpha(car)  
12
```

Compatibilidad con el lenguaje C:  
0 es equivalente a false  
Cualquier valor distinto de 0 es equivalente a true

- Condición compuesta:  
Combinación de condiciones simples y operadores lógicos

```
!isalpha(car)  
(num < 0) || (car == 'a')  
(num < 0) && ((car == 'a') || !isalpha(car))
```

👁️ No confundas el operador de igualdad (==)  
con el operador de asignación (=).



# Evaluación perezosa

## Shortcut Boolean Evaluation

`true || X`  $\equiv$  `true`

`(n == 0) || (x >= 1.0 / n)`

Si n es 0: ¿división por cero? (segunda condición)

Como la primera sería true: ¡no se evalúa la segunda!

`false && X`  $\equiv$  `false`

`(n != 0) && (x < 1.0 / n)`

Si n es 0: ¿división por cero? (segunda condición)

Como la primera sería false: ¡no se evalúa la segunda!

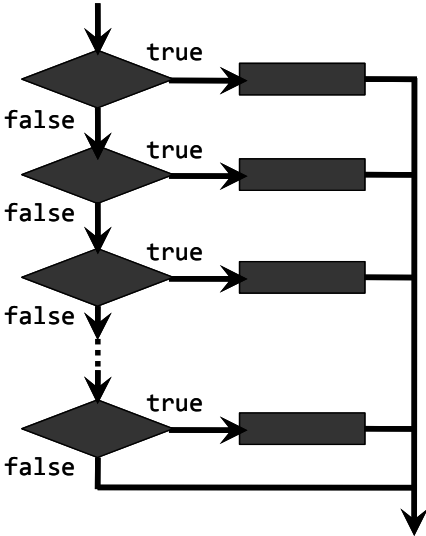


## Selección múltiple

Luis Hernández Yáñez



## Selección múltiple



if-else-if  
switch

Luis Hernández Yáñez



## La escala if-else-if

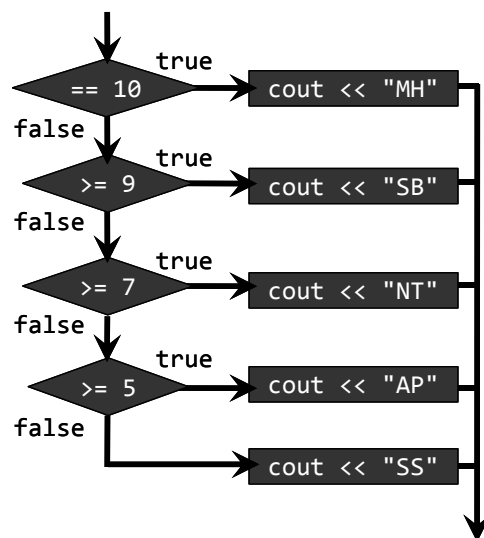
Luis Hernández Yáñez



## La escala if-else-if

Ejemplo:  
Calificación (en letras)  
de un estudiante en base  
a su nota numérica (0-10)

Si nota == 10 entonces MH  
si no, si nota >= 9 entonces SB  
si no, si nota >= 7 entonces NT  
si no, si nota >= 5 entonces AP  
si no SS



Luis Hernández Yáñez



# La escala if-else-if

nota.cpp

```
double nota;
cin >> nota;
if (nota == 10) {
    cout << "MH";
}
else {
    if (nota >= 9) {
        cout << "SB";
    }
    else {
        if (nota >= 7) {
            cout << "NT";
        }
        else {
            if (nota >= 5) {
                cout << "AP";
            }
            else {
                cout << "SS";
            }
        }
    }
}
```

≡

```
double nota;
cin >> nota;
if (nota == 10) {
    cout << "MH";
}
else if (nota >= 9) {
    cout << "SB";
}
else if (nota >= 7) {
    cout << "NT";
}
else if (nota >= 5) {
    cout << "AP";
}
else {
    cout << "SS";
}
```

Luis Hernández Yáñez



# La escala if-else-if

*¡Cuidado con el orden de las condiciones!*

```
double nota;
cin >> nota;
if (nota < 5) { cout << "SS"; }
else if (nota < 7) { cout << "AP"; }
else if (nota < 9) { cout << "NT"; }
else if (nota < 10) { cout << "SB"; }
else { cout << "MH"; }
```



```
double nota;
cin >> nota;
if (nota >= 5) { cout << "AP"; }
else if (nota >= 7) { cout << "NT"; }
else if (nota >= 9) { cout << "SB"; }
else if (nota == 10) { cout << "MH"; }
else { cout << "SS"; }
```

¡No se ejecutan nunca!



Sólo muestra AP o SS

Luis Hernández Yáñez



# La escala if-else-if

## Simplificación de las condiciones

0                      SS                      5                      AP                      7                      NT                      9                      SB                      10

```
if (nota == 10) { cout << "MH"; }
else if ((nota < 10) && (nota >= 9)) { cout << "SB"; }
else if ((nota < 9) && (nota >= 7)) { cout << "NT"; }
else if ((nota < 7) && (nota >= 5)) { cout << "AP"; }
else if (nota < 5) { cout << "SS"; }
```

```
if (nota == 10) { cout << "MH"; }
else if (nota >= 9) { cout << "SB"; }
else if (nota >= 7) { cout << "NT"; }
else if (nota >= 5) { cout << "AP"; }
else { cout << "SS"; }
```

Siempre true: ramas else  
Si no es 10, es menor que 10  
Si no es >= 9, es menor que 9  
Si no es >= 7, es menor que 7  
...  
true && X ≡ X

Luis Hernández Yáñez



# Nivel de un valor

nivel.cpp

```
#include <iostream>
using namespace std;
int main() {
    int num;
    cout << "Introduce el nivel: ";
    cin >> num;
    if (num == 4) {
        cout << "Muy alto" << endl;
    }
    else if (num == 3) {
        cout << "Alto" << endl;
    }
    else if (num == 2) {
        cout << "Medio" << endl;
    }
    else if (num == 1) {
        cout << "Bajo" << endl;
    }
    else {
        cout << "Valor no válido" << endl;
    }
    return 0;
}
```

Si num == 4 entonces Muy alto  
Si num == 3 entonces Alto  
Si num == 2 entonces Medio  
Si num == 1 entonces Bajo

Luis Hernández Yáñez



# ¿Código repetido en las distintas ramas?

---

```
if (num == 4) { cout << "Muy alto" << endl; }
else if (num == 3) { cout << "Alto" << endl; }
else if (num == 2) { cout << "Medio" << endl; }
else if (num == 1) { cout << "Bajo" << endl; }
else cout << "Valor no válido" << endl; }
```



```
if (num == 4) cout << "Muy alto";
else if (num == 3) cout << "Alto";
else if (num == 2) cout << "Medio";
else if (num == 1) cout << "Bajo";
else cout << "Valor no válido";
cout << endl;
```



---

## Fundamentos de la programación

# La instrucción switch



# La instrucción switch

*Selección entre valores posibles de una expresión*

```
switch (expresión) {  
  case constante1:  
    {  
      código1  
    }  
    [break;]  
  case constante2:  
    {  
      código2  
    }  
    [break;]  
  ...  
}
```

→

```
case constanteN:  
{  
  códigoN  
}  
[break;]  
[default:  
{  
  códigoDefault  
}]  
}
```

Luis Hernández Yáñez



# La instrucción switch

nivel2.cpp

```
switch (num) {  
  case 4:  
    {  
      cout << "Muy alto";  
    }  
    break;  
  case 3:  
    {  
      cout << "Alto";  
    }  
    break;  
  case 2:  
    {  
      cout << "Medio";  
    }  
    break;  
  case 1:  
    {  
      cout << "Bajo";  
    }  
    break;  
  default:  
    {  
      cout << "Valor no válido";  
    }  
}
```

Si num == 4 → Muy alto  
Si num == 3 → Alto  
Si num == 2 → Medio  
Si num == 1 → Bajo

Luis Hernández Yáñez



# La instrucción break

*Interrumpe* el switch; continúa en la instrucción que le siga

```
switch (num) {  
  ...  
  case ③  
  {  
    cout << "Alto";  
  }  
  break;  
  case 2:  
  {  
    cout << "Medio";  
  }  
  break;  
  ...  
}
```



Luis Hernández Yáñez



# La instrucción break

```
switch (num) {  
  ...  
  case ③  
  {  
    cout << "Alto";  
  }  
  case 2:  
  {  
    cout << "Medio";  
  }  
  case 1:  
  {  
    cout << "Bajo";  
  }  
  default:  
  {  
    cout << "Valor no válido";  
  }  
}
```

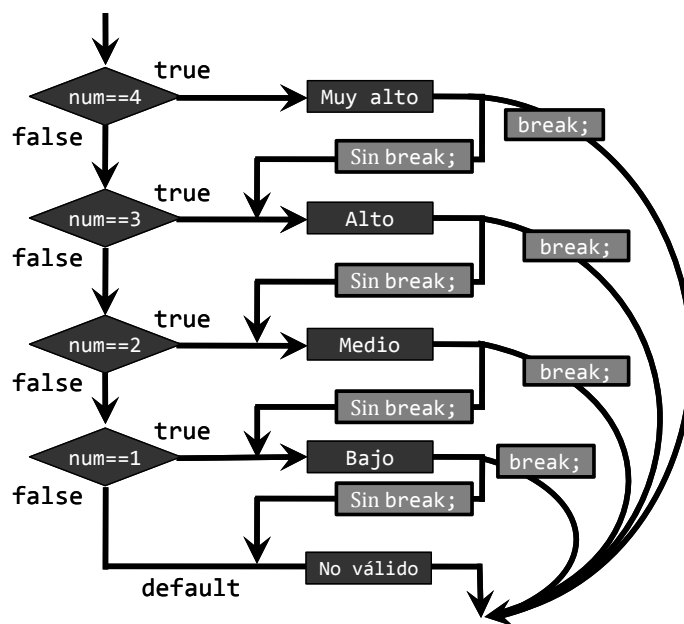


Luis Hernández Yáñez





# Con y sin break



Luis Hernández Yáñez



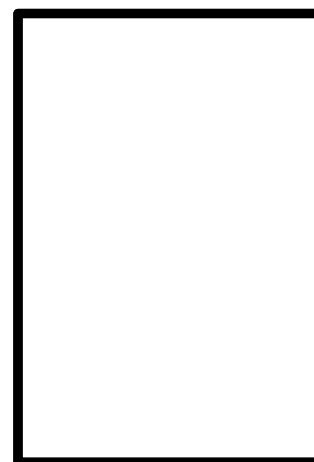
# Un menú

```
int menu() {
    int op = -1; // Cualquiera no válida

    while ((op < 0) || (op > 4)) {
        cout << "1 - Nuevo cliente" << endl;
        cout << "2 - Editar cliente" << endl;
        cout << "3 - Baja cliente" << endl;
        cout << "4 - Ver cliente" << endl;
        cout << "0 - Salir" << endl;
        cout << "Opción: ";
        cin >> op;

        if ((op < 0) || (op > 4)) {
            cout << "¡Opción no válida!" << endl;
        }
    }

    return op;
}
```



Luis Hernández Yáñez



# Un menú

---

```
int opcion;
...
opcion = menu();
switch (opcion) {
case 1:
    {
        cout << "En la opción 1..." << endl;
    }
    break;
case 2:
    {
        cout << "En la opción 2..." << endl;
    }
    break;
case 3:
    {
        cout << "En la opción 3..." << endl;
    }
    break;
case 4:
    {
        cout << "En la opción 4..." << endl;
    } // En la última no necesitamos break
}
```

Luis Hernández Yáñez



# El menú con su bucle...

---

```
int opcion;
...
opcion = menu();
while (opcion != 0) {
    switch (opcion) {
    case 1:
        {
            cout << "En la opción 1..." << endl;
        }
        break;
    case 4:
        {
            cout << "En la opción 4..." << endl;
        }
    } // switch
    ...
    opcion = menu();
} // while
```

Luis Hernández Yáñez



```
int nota; // Sin decimales
cout << "Nota (0-10): ";
cin >> nota;
switch (nota) {
case 0:
case 1:
case 2:
case 3:
case 4:
    {
        cout << "Suspenso";
    }
    break; // De 0 a 4: SS
case 5:
case 6:
    {
        cout << "Aprobado";
    }
    break; // 5 o 6: AP
case 7:
case 8:
    {
        cout << "Notable";
    }
    break; // 7 u 8: NT
case 9:
case 10:
    {
        cout << "Sobresaliente";
    }
    break; // 9 o 10: SB
default:
    {
        cout << "¡No válida!";
    }
}
```



## Escritura de variables de tipos enumerados

```
typedef enum { enero, febrero, marzo, abril, mayo, junio,
             julio, agosto, septiembre, octubre, noviembre, diciembre }
tMes;
tMes mes;
...
switch (mes) {
case enero:
    {
        cout << "enero";
    }
    break;
case febrero:
    {
        cout << "febrero";
    }
    break;
...
case diciembre:
    {
        cout << "diciembre";
    }
}
```

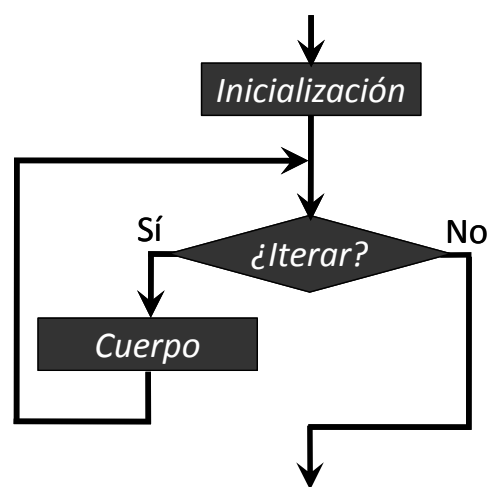


## Repetición



## Repetición (iteración)

---



Bucles while y for



# Tipos de bucles

---

- ✓ Número de iteraciones condicionado (*recorrido variable*):
  - Bucle while  
`while (condición) cuerpo`  
Ejecuta el *cuerpo* mientras la *condición* sea true
  - Bucle do-while  
Comprueba la condición al final (lo veremos más adelante)
- ✓ Número de iteraciones prefijado (*recorrido fijo*):
  - Bucle for  
`for (inicialización; condición; paso) cuerpo`  
Ejecuta el *cuerpo* mientras la *condición* sea true  
Se usa una variable contadora entera



# Fundamentos de la programación

---

## El bucle while



*Mientras la condición sea cierta, ejecuta el cuerpo*

```
while (condición) {  
    cuerpo  
}
```

Condición al principio del bucle

```
int i = 1; // Inicialización de la variable i  
while (i <= 100) {  
    cout << i << endl;  
    i++;  
}
```

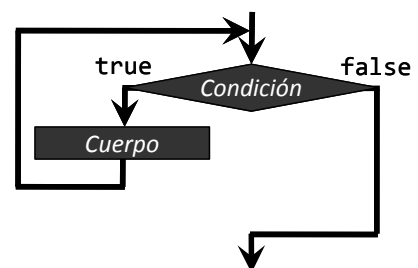
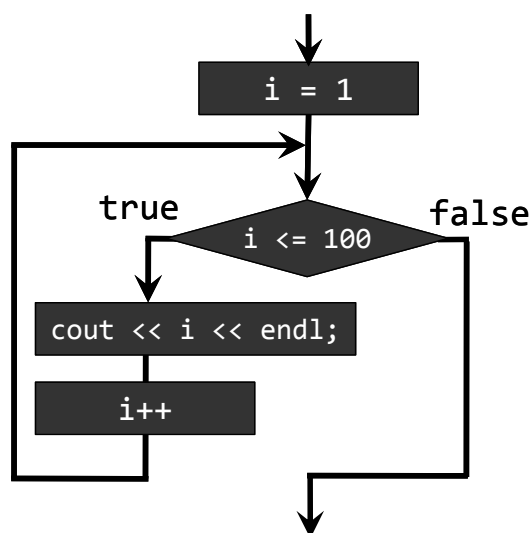
Muestra los números del 1 al 100



## Ejecución del bucle while

```
int i = 1;  
while (i <= 100) {  
    cout << i << endl;  
    i++;  
}
```

i 101



1
2
3
...
99
100
—



# El bucle while

*¿Y si la condición es falsa al comenzar?*

No se ejecuta el cuerpo del bucle ninguna vez

```
int op;
cout << "Introduce la opción: ";
cin >> op;
while ((op < 0) || (op > 4)) {
    cout << "¡No válida! Inténtalo otra vez" << endl;
    cout << "Introduce la opción: ";
    cin >> op;
}
```

Si el usuario introduce un número entre 0 y 4:

No se ejecuta el cuerpo del bucle



## Ejemplo de bucle while

primero.cpp

*Primer entero cuyo cuadrado es mayor que 1.000*

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

*¡Ejecuta el programa para saber cuál es ese número!*

```
int main() {
    int num = 1;
    while (num * num <= 1000) {
        num++;
    }
    cout << "1er. entero con cuadrado mayor que 1.000: "
         << num << endl;
    return 0;
}
```

← Empezamos en 1

← Incrementamos en 1

Recorre la *secuencia* de números 1, 2, 3, 4, 5, ...



```
#include <iostream>
using namespace std;
int main() {
    double num, suma = 0, media = 0;
    int cont = 0;
    cout << "Introduce un número (0 para terminar): ";
    cin >> num;
    while (num != 0) { // 0 para terminar
        suma = suma + num;
        cont++;
        cout << "Introduce un número (0 para terminar): ";
        cin >> num;
    }
    if (cont > 0) {
        media = suma / cont;
    }
    cout << "Suma = " << suma << endl;
    cout << "Media = " << media << endl;
    return 0;
}
```

Recorre la *secuencia* de números introducidos

← Leemos el primero

← Leemos el siguiente

Luis Hernández Yáñez



## Fundamentos de la programación

# El bucle for

Luis Hernández Yáñez





# Bucle for

## *Número de iteraciones prefijado*

Variable contadora que determina el número de iteraciones:

```
for ([int] var = ini; condición; paso) cuerpo
```

La *condición* compara el valor de *var* con un valor final

El *paso* incrementa o decrementa el valor de *var*

El valor de *var* debe ir aproximándose al valor final

```
for (int i = 1; i <= 100; i++)... 1, 2, 3, 4, 5, ..., 100
```

```
for (int i = 100; i >= 1; i--)... 100, 99, 98, 97, ..., 1
```

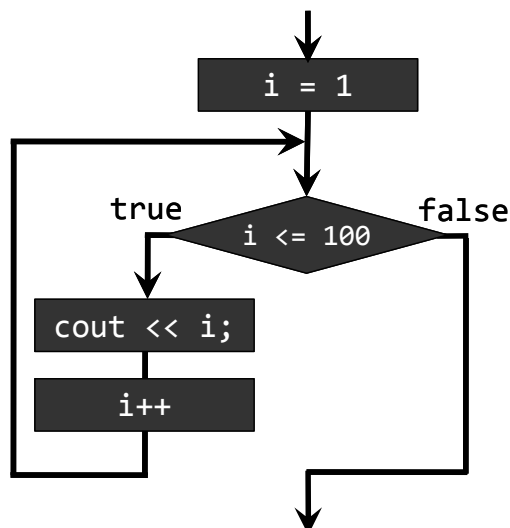
Tantos ciclos como valores toma la variable contadora



# Ejecución del bucle for

```
for (inicialización; condición; paso) cuerpo
```

```
for (int i = 1; i <= 100; i++) {  
    cout << i;  
}
```

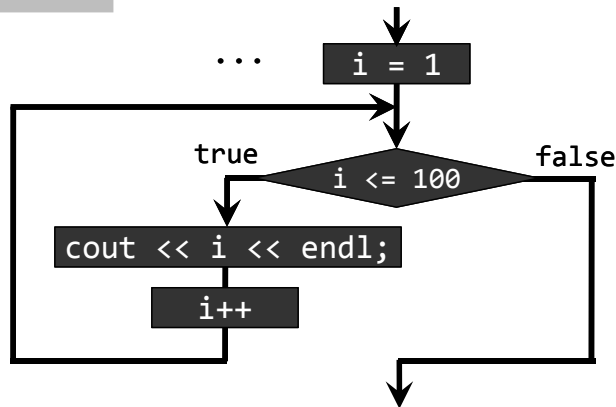


# Ejecución del bucle for

for1.cpp

```
for (int i = 1; i <= 100; i++) {  
    cout << i << endl;  
}
```

i 101



1
2
3
...
99
100
—

Luis Hernández Yáñez



# Bucle for

for2.cpp

## La variable contadora

El *paso* no tiene porqué ir de uno en uno:

```
for (int i = 1; i <= 100; i = i + 2)  
    cout << i << endl;
```

Este bucle for muestra los números impares de 1 a 99



*Muy importante*

El cuerpo del bucle **NUNCA** debe alterar el valor del contador

## Garantía de terminación

Todo bucle debe terminar su ejecución

Bucles for: la variable contadora debe converger al valor final

Luis Hernández Yáñez



# Ejemplo de bucle for

suma.cpp

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

long long int suma(int n);

int main() {
    int num;
    cout << "Número final: ";
    cin >> num;
    if (num > 0) { // El número debe ser positivo
        cout << "La suma de los números entre 1 y "
            << num << " es: " << suma(num);
    }
    return 0;
}

long long int suma(int n) {
    long long int total = 0;
    for (int i = 1; i <= n; i++) {
        total = total + i;
    }
    return total;
}
```

$$\sum_{i=1}^N i$$

Recorre la *secuencia* de números  
1, 2, 3, 4, 5, ..., n

Luis Hernández Yáñez



## Bucle for

*¿Incremento/decremento prefijo o postfijo?*

Es indiferente

Estos dos bucles producen el mismo resultado:

```
for (int i = 1; i <= 100; i++) ...
```

```
for (int i = 1; i <= 100; ++i) ...
```

*Bucles infinitos*

```
for (int i = 1; i <= 100; i--) ...
```

1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 ...

Cada vez más lejos del valor final (100)

Es un error de diseño/programación

Luis Hernández Yáñez



# Ámbito de la variable contadora

---

## *Declarada en el propio bucle*

```
for (int i = 1; ...)
```

Sólo se conoce en el cuerpo del bucle (su ámbito)

No se puede usar en instrucciones que sigan al bucle

## *Declarada antes del bucle*

```
int i;
```

```
for (i = 1; ...)
```

Se conoce en el cuerpo del bucle y después del mismo

Ámbito externo al bucle



# Bucle for versus bucle while

---

Los bucles for se pueden reescribir como bucles condicionados

```
for (int i = 1; i <= 100; i++) cuerpo
```

Es equivalente a:

```
int i = 1;
while (i <= 100) {
    cuerpo
    i++;
}
```

La inversa no es siempre posible:

```
int i;
cin >> i;
while (i != 0) {
    cuerpo
    cin >> i;
}
```

*¿Bucle for equivalente?  
¡No sabemos cuántos números  
introducirá el usuario!*



## Bucles anidados



## Bucles for anidados

---

Un bucle for en el cuerpo de otro bucle for

Cada uno con su propia variable contadora:

```
for (int i = 1; i <= 100; i++) {  
    for (int j = 1; j <= 5; j++) {  
        cuerpo  
    }  
}
```

Para cada valor de *i*  
el valor de *j* varía entre 1 y 5

*j* varía más rápido que *i*

<i>i</i>	<i>j</i>
1	1
1	2
1	3
1	4
1	5
2	1
2	2
2	3
2	4
2	5
3	1

...



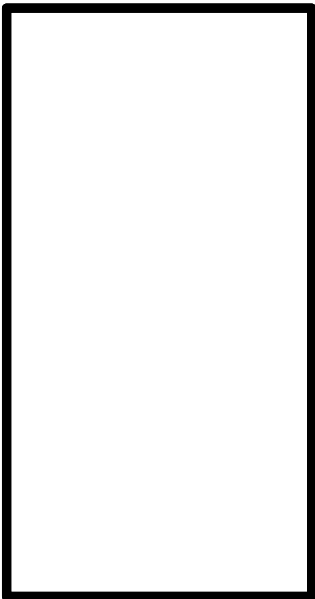
# Tablas de multiplicación

tablas.cpp

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

int main() {
    for (int i = 1; i <= 10; i++) {
        for (int j = 1; j <= 10; j++) {
            cout << setw(2) << i << " x "
                << setw(2) << j << " = "
                << setw(3) << i * j << endl;
        }
    }

    return 0;
}
```



Luis Hernández Yáñez



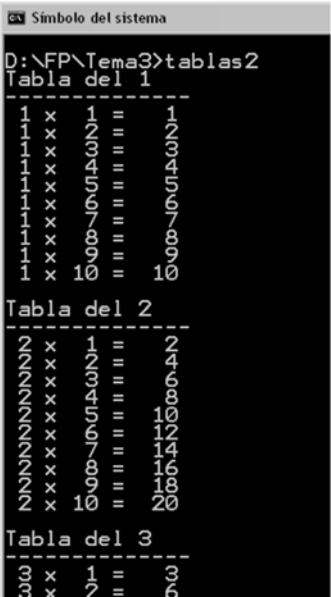
# Mejor presentación

tablas2.cpp

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

int main() {
    for (int i = 1; i <= 10; i++) {
        cout << "Tabla del " << i << endl;
        cout << "-----" << endl;
        for (int j = 1; j <= 10; j++) {
            cout << setw(2) << i << " x "
                << setw(2) << j << " = "
                << setw(3) << i * j << endl;
        }
        cout << endl;
    }

    return 0;
}
```



Luis Hernández Yáñez



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

int menu(); // 1: Tablas de multiplicación; 2: Sumatorio
long long int suma(int n); // Sumatorio

int main() {
    int opcion = menu();
    while (opcion != 0) {
        switch (opcion) {
            case 1:
                {
                    for (int i = 1; i <= 10; i++) {
                        for (int j = 1; j <= 10; j++) {
                            cout << setw(2) << i << " x "
                                << setw(2) << j << " = "
                                << setw(3) << i * j << endl;
                        }
                    }
                }
            break; ...
        }
    }
}
```



# Más bucles anidados

```
case 2:
{
    int num = 0;
    while (num <= 0) {
        cout << "Hasta (positivo)? ";
        cin >> num;
    }
    cout << "La suma de los números del 1 al "
        << num << " es: " << suma(num) << endl;
}
} // switch
opcion = menu();
} // while (opcion != 0)
return 0;
}
```



# Más bucles anidados

```
int menu() {
    int op = -1;
    while ((op < 0) || (op > 2)) {
        cout << "1 - Tablas de multiplicar" << endl;
        cout << "2 - Sumatorio" << endl;
        cout << "0 - Salir" << endl;
        cout << "Opción: " << endl;
        cin >> op;
        if ((op < 0) || (op > 2)) {
            cout << "¡Opción no válida!" << endl;
        }
    }
    return op;
}

long long int suma(int n) {
    long long int total = 0;
    for (int i = 1; i <= n; i++) {
        total = total + i;
    }
    return total;
}
```

Luis Hernández Yáñez



# Ambos tipos de bucles anidados

```
while (opcion != 0) {
    ...
    for (int i = 1; i <= 10; i++) {
        for (int j = 1; j <= 10; j++) {
            ...
        }
    }
    while (num <= 0) {
        ...
    }
    suma()
    for (int i = 1; i <= n; i++) {
        ...
    }
    while ((op < 0) || (op > 2)) {
        ...
    }
}
menu()
```

Luis Hernández Yáñez





## Ámbito y visibilidad



## Ámbito de los identificadores

---

Cada bloque crea un nuevo ámbito:

```
int main() {  
    double d = -1, suma = 0;           3 ámbitos anidados  
    int cont = 0;  
    while (d != 0) {  
        cin >> d;  
        if (d != 0) {  
            suma = suma + d;  
            cont++;  
        }  
    }  
    cout << "Suma = " << suma << endl;  
    cout << "Media = " << suma / cont << endl;  
    return 0;  
}
```



# Ámbito de los identificadores

---

Un identificador se conoce  
en el ámbito en el que está declarado  
(a partir de su instrucción de declaración)  
y en los subámbitos posteriores



# Ámbito de los identificadores

---

```
int main() {  
    double d;           Ámbito de la variable d  
    if (...) {  
        int cont = 0;  
        for (int i = 0; i <= 10; i++) {  
            ...  
        }  
    }  
    char c;  
    if (...) {  
        double x;  
        ...  
    }  
    return 0;  
}
```



# Ámbito de los identificadores

---

```
int main() {
    double d;
    if (...) {
        int cont = 0;    Ámbito de la variable cont
        for (int i = 0; i <= 10; i++) {
            ...
        }
    }
    char c;
    if (...) {
        double x;
        ...
    }
    return 0;
}
```



# Ámbito de los identificadores

---

```
int main() {
    double d;
    if (...) {
        int cont = 0;
        for (int i = 0; i <= 10; i++) {
            ...
        }
    }
    char c;
    if (...) {
        double x;
        ...
    }
    return 0;
}
```



# Ámbito de los identificadores

---

```
int main() {
    double d;
    if (...) {
        int cont = 0;
        for (int i = 0; i <= 10; i++) {
            ...
        }
    }
    char c;
    if (...) {
        double x;
        ...
    }
    return 0;
}
```

Ámbito de la variable c

Luis Hernández Yáñez



# Ámbito de los identificadores

---

```
int main() {
    double d;
    if (...) {
        int cont = 0;
        for (int i = 0; i <= 10; i++) {
            ...
        }
    }
    char c;
    if (...) {
        double x;
        ...
    }
    return 0;
}
```

Ámbito de la variable x

Luis Hernández Yáñez



# Visibilidad de los identificadores

Si en un subámbito se declara un identificador con idéntico nombre que uno ya declarado en el ámbito, el del subámbito *oculta* al del ámbito (no es visible)



# Visibilidad de los identificadores

```
int main() {  
    int i, x;           Oculta, en su ámbito, a la i anterior  
    if (...) {  
        int i = 0;     Oculta, en su ámbito, a la i anterior  
        for(int i = 0; i <= 10; i++) {  
            ...  
        }  
    }  
    char c;  
    if (...) {  
        double x;     Oculta, en su ámbito, a la x anterior  
        ...  
    }  
    return 0;  
}
```



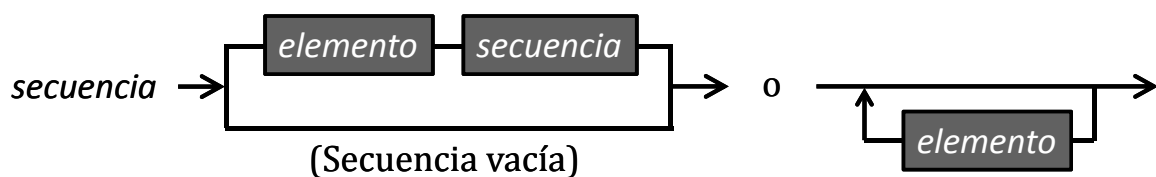
## Secuencias



## Secuencias



Sucesión de elementos de un mismo tipo que se acceden linealmente



1 34 12 26 4 87 184 52

Comienza en un *primer* elemento (si no está vacía)

A cada elemento le sigue otra secuencia (vacía, si es el *último*)

*Acceso secuencial (lineal)*

Se comienza siempre accediendo al primer elemento

Desde un elemento sólo se puede acceder a su elemento siguiente (*sucesor*), si es que existe

Todos los elementos, de un mismo tipo



# Secuencias en programación

No tratamos secuencias infinitas: siempre hay un último elemento

- ✓ Secuencias explícitas:
  - Sucesión de datos de un dispositivo (teclado, disco, sensor, ...)
- ✓ Secuencias calculadas:
  - Fórmula de recurrencia que determina el elemento siguiente
- ✓ Listas (*más adelante*)

Secuencias explícitas que manejaremos:

Datos introducidos por el teclado o leídos de un archivo

Con un elemento especial al final de la secuencia (*centinela*)

1 34 12 26 4 87 184 52 -1



## Detección del final de la secuencia

- ✓ Secuencia explícita leída de archivo:
  - Detectar la marca de final de archivo (Eof - *End of file*)
  - Detectar un valor centinela al final ←
- ✓ Secuencia explícita leída del teclado:
  - Preguntar al usuario si quiere introducir un nuevo dato
  - Preguntar al usuario primero cuántos datos va a introducir
  - Detectar un valor centinela al final ←

Valor *centinela*:

Valor especial al final que no puede darse en la secuencia

(Secuencia de números positivos → centinela: cualquier negativo)

12 4 37 23 8 19 83 63 2 35 17 76 15 -1



# Centinelas

Debe haber algún valor que no sea un elemento válido

Secuencias numéricas:

Si se permite cualquier número, no hay centinela posible

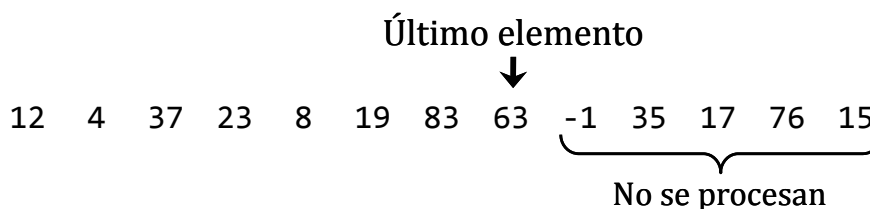
Cadenas de caracteres:

¿Caracteres especiales (no imprimibles)?

En realidad el valor centinela es parte de la secuencia, pero su significado es especial y no se procesa como el resto

Significa que se ha alcanzado el final de la secuencia

*(Incluso aunque haya elementos posteriores)*



# Esquemas de tratamiento de secuencias

Tratamiento de los elementos uno a uno desde el primero

## *Recorrido*

Un mismo tratamiento para todos los elementos de la secuencia

Ej.- Mostrar los elementos de una secuencia, sumar los números de una secuencia, ¿par o impar cada número de una secuencia?, ...

Termina al llegar al final de la secuencia

## *Búsqueda*

Recorrido de la secuencia hasta encontrar un elemento buscado

Ej.- Localizar el primer número que sea mayor que 1.000

Termina al localizar el primer elemento que cumple la condición o al llegar al final de la secuencia (*no encontrado*)





## Recorrido de secuencias



## Esquema de recorrido

---

*Un mismo tratamiento a todos los elementos*

*Inicialización*

*Mientras no se llegue al final de la secuencia:*

*Obtener el siguiente elemento*

*Procesar el elemento*

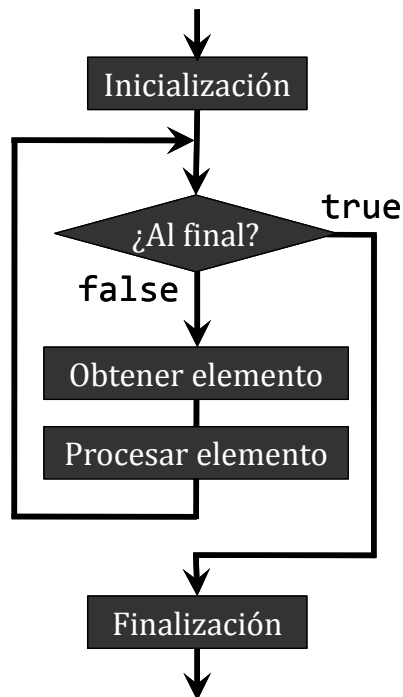
*Finalización*

Al empezar se obtiene el primer elemento de la secuencia

En los siguientes pasos del bucle se van obteniendo los siguientes elementos de la secuencia



# Esquema de recorrido



No sabemos cuántos elementos hay  
→ No podemos implementar con for



# Secuencias explícitas con centinela

## Implementación con while

*Iniciación*

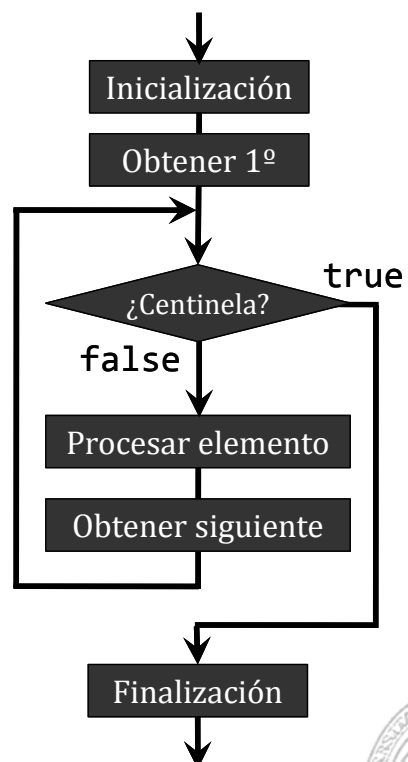
*Obtener el primer elemento*

*Mientras no sea el centinela:*

*Procesar el elemento*

*Obtener el siguiente elemento*

*Finalización*



# Secuencias explícitas leídas del teclado

## *Secuencia de números positivos*

Siempre se realiza al menos una lectura

Centinela: -1

```
double d, suma = 0; _____ Inicialización
cout << "Valor (-1 termina): ";
cin >> d; } Primer elemento
while (d != -1) { _____ Mientras no el centinela
    suma = suma + d; _____ Procesar elemento
    cout << "Valor (-1 termina): ";
    cin >> d; } Siguiete elemento
}
cout << "Suma = " << suma << endl; — Finalización
```



# Secuencias explícitas leídas del teclado

## *Longitud de una secuencia de caracteres*

`longitud.cpp`

Centinela: carácter punto (.)

```
int longitud() {
    int l = 0;
    char c;
    cout << "Texto terminado en punto: ";
    cin >> c; // Obtener primer carácter
    while (c != '.') { // Mientras no el centinela
        l++; // Procesar
        cin >> c; // Obtener siguiente carácter
    }
    return l;
}
```



# Secuencias explícitas leídas del teclado

*¿Cuántas veces aparece un carácter en una cadena?*

Centinela: asterisco (\*)

cont.cpp

```
char buscado, c;
int cont = 0;
cout << "Carácter a buscar: ";
cin >> buscado;
cout << "Cadena: ";
cin >> c;
while (c != '*') {
    if (c == buscado) {
        cont++;
    }
    cin >> c;
}
cout << buscado << " aparece " << cont
    << " veces.";
```

— Primer elemento  
— Mientras no el centinela  
} Procesar elemento  
— Siguiente elemento

Luis Hernández Yáñez



# Secuencias explícitas leídas de archivo

*Suma de los números de la secuencia*

suma2.cpp

Centinela: 0

```
int sumaSecuencia() {
    double d, suma = 0;
    ifstream archivo; // Archivo de entrada (lectura)
    archivo.open("datos.txt");
    if (archivo.is_open()) {
        archivo >> d; // Obtener el primero
        while (d != 0) { // Mientras no sea el centinela
            suma = suma + d; // Procesar el dato
            archivo >> d; // Obtener el siguiente
        }
        archivo.close();
    }
    return suma;
}
```

Luis Hernández Yáñez



## Secuencias calculadas

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

Página 363



## Secuencias calculadas

sumatorio.cpp

Recurrencia:  $e_{i+1} = e_i + 1$        $e_1 = 1$

1 2 3 4 5 6 7 8 ...

Suma de los números de la secuencia calculada:

$$\sum_{i=1}^N i$$

```
int main() {
    int num;
    cout << "N = ";
    cin >> num;
    cout << "Sumatorio:" << suma(num);
    return 0;
}
long long int suma(int n) {
    int sumatorio = 0;
    for (int i = 1; i <= n; i++) {
        sumatorio = sumatorio + i;
    }
    return sumatorio;
}
```

Último elemento de la secuencia: n

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

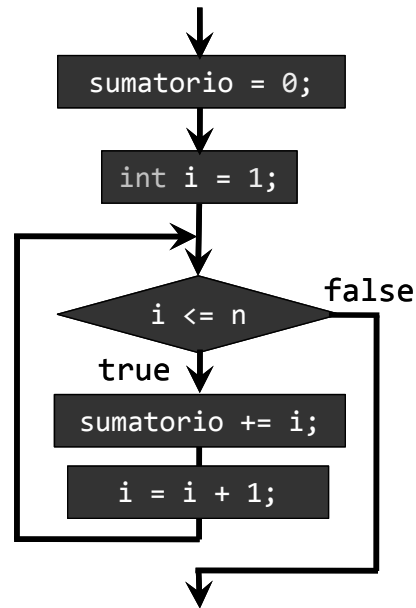
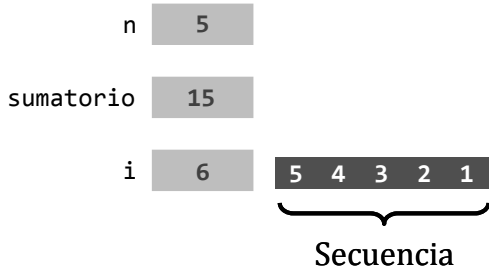
Página 364



# Suma de una secuencia calculada

```
long long int suma(int n) {
    int sumatorio = 0;
    for (int i = 1; i <= n; i++) {
        sumatorio = sumatorio + i;
    }
    ...
}
```

$$\sum_{i=1}^N i$$



Luis Hernández Yáñez



# Números de Fibonacci

## Definición

$$F_i = F_{i-1} + F_{i-2}$$

$$F_1 = 0$$

$$F_2 = 1$$

0 1 1 2 3 5 8 13 21 34 55 89 ...

## ¿Fin de la secuencia?

Primer número de Fibonacci mayor que un número dado

Ese número de Fibonacci actúa como centinela

Si num es 50, la secuencia será:

0 1 1 2 3 5 8 13 21 34

Luis Hernández Yáñez



## Recorrido de la secuencia calculada

```
int num, fib, fibMenos2 = 0, fibMenos1 = 1; // 1º y 2º
fib = fibMenos2 + fibMenos1; // Calculamos el tercero
cout << "Hasta: ";
cin >> num;
if (num >= 1) { // Ha de ser entero positivo
    cout << "0 1 "; // Los dos primeros son <= num
    while (fib <= num) { // Mientras no mayor que num
        cout << fib << " ";
        fibMenos2 = fibMenos1; // Actualizamos anteriores
        fibMenos1 = fib; // para obtener...
        fib = fibMenos2 + fibMenos1; // ... el siguiente
    }
}
```



*¿Demasiados comentarios?*

Para no oscurecer el código, mejor una explicación al principio



# Números de Fibonacci

El bucle calcula adecuadamente la secuencia:

```
→ while (fib <= num) {
→     cout << fib << " ";
→     fibMenos2 = fibMenos1;
→     fibMenos1 = fib;
→     fib = fibMenos2 + fibMenos1;
→ }
```

num 100

0 1 1 2 3 5 ...

fib 5

fibMenos1 3

fibMenos2 2



## Búsqueda en secuencias



## Esquema de búsqueda

---

### *Localización del primer elemento con una propiedad*

*Inicialización*

*Mientras no se encuentre el elemento  
y no se esté al final de la secuencia:*

*Obtener el siguiente elemento*

*Comprobar si el elemento satisface la condición*

*Finalización*

*(tratar el elemento encontrado o indicar que no se ha encontrado)*

Elemento que se busca: satisfará una condición

Dos condiciones de terminación del bucle: se encuentra / al final

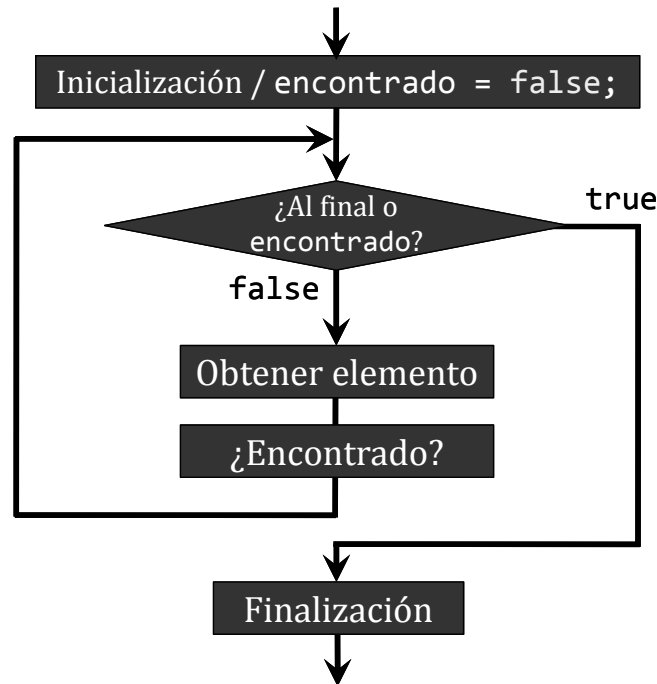
Variable lógica que indique si se ha encontrado





# Esquema de búsqueda

## Localización del primer elemento con una propiedad



Luis Hernández Yáñez



# Secuencias explícitas con centinela

## Implementación con while

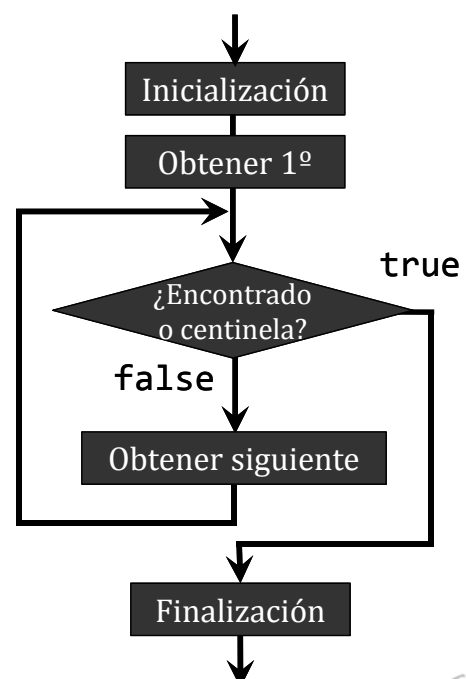
*Inicialización*

*Obtener el primer elemento*

*Mientras ni encontrado ni el centinela:*

*Obtener el siguiente elemento*

*Finalización (¿encontrado?)*



Luis Hernández Yáñez



# Secuencias explícitas leídas del teclado

*Primer número mayor que uno dado*

busca.cpp

```
Centinela: -1
double d, num;
bool encontrado = false;
cout << "Encontrar primero mayor que: ";
cin >> num;
cout << "Siguiete (-1 para terminar): ";
cin >> d; // Obtener el primer elemento
while ((d != -1) && !encontrado) {
    // Mientras no sea el centinela y no se encuentre
    if (d > num) { // ¿Encontrado?
        encontrado = true;
    }
    else {
        cout << "Siguiete (-1 para terminar): ";
        cin >> d; // Obtener el siguiente elemento
    }
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

Página 373



## Fundamentos de la programación

# Arrays de tipos simples

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

Página 374



# Arrays

## Colecciones homogéneas

Un mismo tipo de dato para varios elementos:

- ✓ Notas de los estudiantes de una clase
- ✓ Ventas de cada día de la semana
- ✓ Temperaturas de cada día del mes

...

En lugar de declarar  $N$  variables...

vLun	vMar	vMie	vJue	vVie	vSab	vDom
125.40	76.95	328.80	254.62	435.00	164.29	0.00

... declaramos una tabla de  $N$  valores:

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
Índices →	0	1	2	3	4	5	6

Luis Hernández Yáñez



# Arrays

## Estructura secuencial

Cada elemento se encuentra en una posición (*índice*):

- ✓ Los índices son enteros positivos
- ✓ El índice del primer elemento siempre es 0
- ✓ Los índices se incrementan de uno en uno

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
	0	1	2	3	4	5	6

### Acceso directo

A cada elemento se accede a través de su índice:

`ventas[4]` accede al 5º elemento (contiene el valor **435.00**)

`cout << ventas[4];`

`ventas[4] = 442.75;`



Datos de un mismo tipo base:  
Se usan como cualquier variable

Luis Hernández Yáñez




# Tipos arrays

## Declaración de tipos de arrays

```
typedef tipo_base nombre_tipo[tamaño];
```

Ejemplos:

```
typedef double tTemp[7];  
typedef short int tDiasMes[12];  
typedef char tVocales[5];  
typedef double tVentas[31];  
typedef tMoneda tCalderilla[15]; // Enumerado tMoneda
```

 **Recuerda:** Adoptamos el convenio de comenzar los nombres de tipo con una *t* minúscula, seguida de una o varias palabras, cada una con su inicial en mayúscula



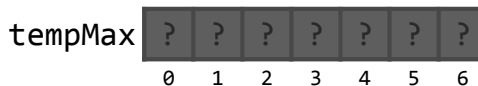
# Variables arrays

## Declaración de variables arrays

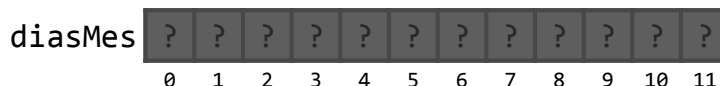
*tipo nombre;*

Ejemplos:

```
tTemp tempMax;
```



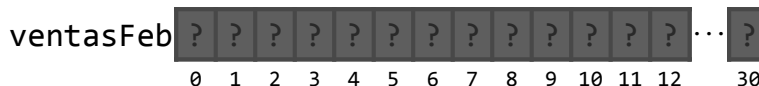
```
tDiasMes diasMes;
```



```
tVocales vocales;
```



```
tVentas ventasFeb;
```



 **NO se inicializan los elementos automáticamente**



## Uso de variables arrays



## Acceso a los elementos de un array

---

*nombre*[índice]

Cada elemento se accede a través de su índice (posición en el array)

tVocales vocales;

```
typedef char tVocales[5];
```

vocales	'a'	'e'	'i'	'o'	'u'
	0	1	2	3	4

5 elementos, índices de 0 a 4:

vocales[0]    vocales[1]    vocales[2]    vocales[3]    vocales[4]

Procesamiento de cada elemento:

Como cualquier otra variable del tipo base

```
cout << vocales[4];
```

```
vocales[3] = 'o';
```

```
if (vocales[i] == 'e') ...
```



# Acceso a los elementos de un array

---

¡IMPORTANTE!

¡No se comprueba si el índice es correcto!

*¡Es responsabilidad del programador!*

```
const int Dim = 100;  
typedef double tVentas[Dim];  
tVentas ventas;
```

Índices válidos: enteros entre 0 y Dim-1

ventas[0] ventas[1] ventas[2] ... ventas[98] ventas[99]

¿Qué es ventas[100]? ¿0 ventas[-1]? ¿0 ventas[132]?

¡Memoria de alguna otra variable del programa!



Define los tamaños de los arrays con constantes



# Fundamentos de la programación

---

## Recorrido de arrays



# Recorrido de arrays

Arrays: tamaño fijo → Bucle de recorrido fijo (for)

Ejemplo: Media de un array de temperaturas

```
const int Dias = 7;
typedef double tTemp[Dias];
tTemp temp;
double media, total = 0;
...
for (int i = 0; i < Dias; i++) {
    total = total + temp[i];
}
media = total / Dias;
```

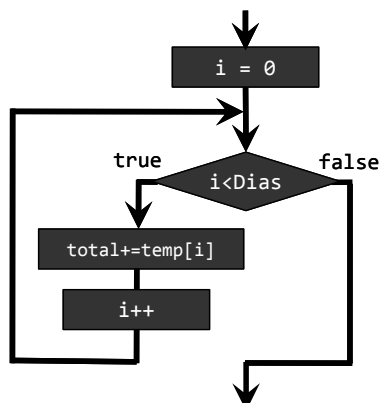
Luis Hernández Yáñez



# Recorrido de arrays

12.40	10.96	8.43	11.65	13.70	13.41	14.07
0	1	2	3	4	5	6

```
tTemp temp;
double media, total = 0;
...
for (int i = 0; i < Dias; i++) {
    total = total + temp[i];
}
```



	Memoria
Dias	7
temp[0]	12.40
temp[1]	10.96
temp[2]	8.43
temp[3]	11.65
temp[4]	13.70
temp[5]	13.41
temp[6]	14.07
media	?
total	84.62
i	7

Luis Hernández Yáñez



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

const int Dias = 7;
typedef double tTemp[Dias];

double media(const tTemp temp);

int main() {
    tTemp temp;
    for (int i = 0; i < Dias; i++) { // Recorrido del array
        cout << "Temperatura del día " << i + 1 << ": ";
        cin >> temp[i];
    }
    cout << "Temperatura media: " << media(temp) << endl;
    return 0;
}
...
```

Los usuarios usan de 1 a 7 para numerar los días  
La interfaz debe aproximarse a los usuarios,  
aunque internamente se usen los índices de 0 a 6

Luis Hernández Yáñez



# Recorrido de arrays

```
double media(const tTemp temp) {
    double med, total = 0;

    for (int i = 0; i < Dias; i++) { // Recorrido del array
        total = total + temp[i];
    }
    med = total / Dias;

    return med;
}
```



Los arrays se pasan a las funciones como constantes  
Las funciones no pueden devolver arrays

Luis Hernández Yáñez





# Arrays de tipos enumerados

---

```
const int Cuantas = 15;
typedef enum { centimo, dos_centimos, cinco_centimos,
             diez_centimos, veinte_centimos, medio_euro, euro } tMoneda;
typedef tMoneda tCalderilla[Cuantas];
string aCadena(tMoneda moneda);
// Devuelve la cadena correspondiente al valor de moneda

tCalderilla bolsillo; // Exactamente llevo Cuantas monedas
bolsillo[0] = euro;
bolsillo[1] = cinco_centimos;
bolsillo[2] = medio_euro;
bolsillo[3] = euro;
bolsillo[4] = centimo;
...
for (int moneda = 0; moneda < Cuantas; moneda++)
    cout << aCadena(bolsillo[moneda]) << endl;
```



# Fundamentos de la programación

---

## Búsqueda en arrays



*¿Qué día las ventas superaron los 1.000 €?*

```
const int Dias = 365; // Año no bisiesto
typedef double tVentas[Dias];

int busca(const tVentas ventas) {
// Índice del primer elemento mayor que 1000 (-1 si no hay)
    bool encontrado = false;
    int ind = 0;
    while ((ind < Dias) && !encontrado) { // Esquema de búsqueda
        if (ventas[ind] > 1000) {
            encontrado = true;
        }
        else {
            ind++;
        }
    }
    if (!encontrado) {
        ind = -1;
    }
    return ind;
}
```

Luis Hernández Yáñez



## Fundamentos de la programación

# Capacidad y copia de arrays

Luis Hernández Yáñez



# Capacidad de los arrays

---

La capacidad de un array no puede ser alterada en la ejecución

El tamaño de un array es una decisión de diseño:

- ✓ En ocasiones será fácil (días de la semana)
- ✓ Cuando pueda variar ha de estimarse un tamaño  
Ni corto ni con mucho desperdicio (posiciones sin usar)

STL (*Standard Template Library*) de C++:

Colecciones más eficientes cuyo tamaño puede variar



# Copia de arrays

---

No se pueden copiar dos arrays (del mismo tipo) con asignación:

```
array2 = array1; // ¡¡¡ NO COPIA LOS ELEMENTOS !!!
```

Han de copiarse los elementos uno a uno:

```
for (int i = 0; i < N; i++) {  
    array2[i] = array1[i];  
}
```



## Arrays no completos



## Arrays no completos

---

Puede que no necesitemos todas las posiciones de un array...

La dimensión del array será el máximo de elementos

Pero podremos tener menos elementos del máximo

Necesitamos un contador de elementos...

```
const int Max = 100;  
typedef double tArray[Max];  
tArray lista;  
int contador = 0;
```

contador: indica cuántas posiciones del array se utilizan

Sólo accederemos a las posiciones entre 0 y contador-1

Las demás posiciones no contienen información del programa



# Arrays no completos

lista.cpp

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

const int Max = 100;
typedef double tArray[Max];

double media(const tArray lista, int cont);

int main() {
    tArray lista;
    int contador = 0;
    double valor, med;
    ifstream archivo;
    archivo.open("lista.txt");
    if (archivo.is_open()) {
        archivo >> valor;
        while ((valor != -1) && (contador < Max)) {
            lista[contador] = valor;
            contador++;
            archivo >> valor;
        } ...
    }
```

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

Página 395



# Arrays no completos

```
        archivo.close();
        med = media(lista, contador);
        cout << "Media de los elementos de la lista: " << med << endl;
    }
    else {
        cout << "¡No se pudo abrir el archivo!" << endl;
    }

    return 0;
}

double media(const tArray lista, int cont) {
    double med, total = 0;
    for (int ind = 0; ind < cont; ind++) {
        total = total + lista[ind];
    }
    med = total / cont;
    return med;
}
```

Sólo recorreremos hasta cont-1

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II

Página 396








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.





ANEXO I

## El operador ternario ?

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## El operador ternario ?



Dos alternativas

– *Condición*: Expresión lógica

– *Exp1* y *Exp2*: Expresiones

Si *Condición* se evalúa a true,  
el resultado es *Exp1*;  
si *Condición* se evalúa a false,  
el resultado es *Exp2*.

```
int a = 5, b = 3, c;  
c = (a + b == 10) ? 2 : 3;  
c = ( 8 == 10) ? 2 : 3;  
c = false ? 2 : 3;  
c = 3;
```

Operadores (prioridad)
++ -- (postfijos)
Llamadas a funciones
Moldes
++ -- (prefijos) !
- (cambio de signo)
* / %
+ -
< <= > >=
== !=
&&
? :
= += -= *= /= %=

Luis Hernández Yáñez



# El operador ternario ?

## Equivalencia con un if-else

```
c = (a + b == 10) ? 2 : 3;
```

Es equivalente a:

```
if (a + b == 10) c = 2;  
else c = 3;
```

Se pueden concatenar:

```
cout << (nota == 10 ? "MH" : (nota >= 9 ? "SB" :  
(nota >= 7 ? "NT" : (nota >= 5 ? "AP" : "SS"))));
```

Esto es equivalente a la escala if-else-if de la siguiente sección.



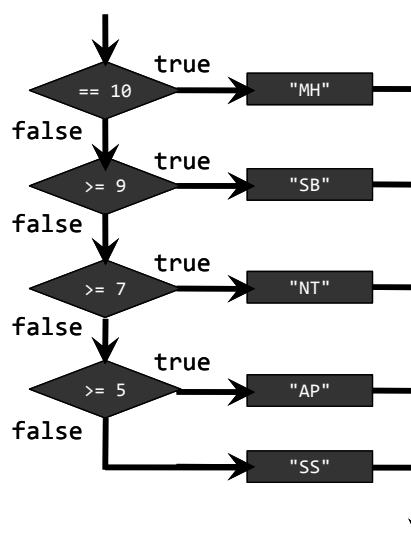
# El operador ternario ?

## Escala if ... else if ... equivalente

```
cout << (nota == 10 ? "MH" : (nota >= 9 ? "SB" :  
(nota >= 7 ? "NT" : (nota >= 5 ? "AP" : "SS"))));
```

Si nota == 10 entonces MH  
si no, si nota >= 9 entonces SB  
si no, si nota >= 7 entonces NT  
si no, si nota >= 5 entonces AP  
si no SS

```
double nota;  
cin >> nota;  
if (nota == 10) { cout << "MH"; }  
else if (nota >= 9) { cout << "SB"; }  
else if (nota >= 7) { cout << "NT"; }  
else if (nota >= 5) { cout << "AP"; }  
else { cout << "SS"; }
```







ANEXO II

## Ejemplos de secuencias

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Recorridos	404
Un aparcamiento	405
¿Paréntesis bien emparejados?	409
¿Dos secuencias iguales?	412
Números primos menores que N	413
Búsquedas	417
Búsqueda de un número en un archivo	419
Búsquedas en secuencias ordenadas	420



## Recorridos

Luis Hernández Yáñez



## Un aparcamiento

---

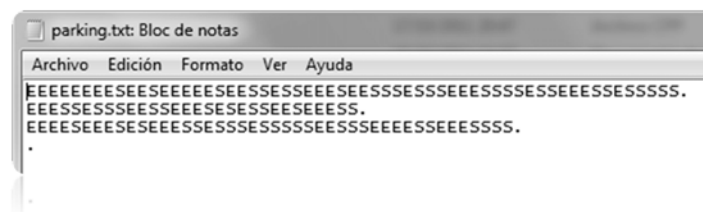
Secuencia de caracteres E y S en archivo

E = Entra un coche; S = Sale un coche

¿Cuántos coches quedan al final de la jornada?

Varios casos, cada uno en una línea y terminado en punto

Final: línea sólo con punto



Luis Hernández Yáñez



# Un aparcamiento

---

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

int main() {
    int coches;
    char c;
    bool terminar = false;
    ifstream archivo;
    archivo.open("parking.txt");
    if (!archivo.is_open()) {
        cout << "¡No se ha podido abrir el archivo!" << endl;
    }
    else {
        // Recorrido...
        archivo.close();
    }
    return 0;
}
```

Luis Hernández Yáñez



# Un aparcamiento (recorrido)

---

```
while (!terminar) {
    archivo >> c;
    if (c == '.') { // . como primer carácter? (centinela)
        terminar = true;
    }
    else {
        coches = 0;
        while (c != '.') { // Recorrido de la secuencia
            cout << c;
            if (c == 'E') {
                coches++;
            }
            else if (c == 'S') {
                coches--;
            }
            archivo >> c;
        }
        ...
    }
}
```

Luis Hernández Yáñez



```
if (coches >= 0) {
    cout << endl << "Quedan " << coches << " coches.";
}
else {
    cout << endl << "Error: Más salidas que entradas!";
}
cout << endl;
}
```



## ¿Paréntesis bien emparejados?

*Cada paréntesis, con su pareja*

Secuencia de caracteres terminada en # y con parejas de paréntesis:

a b ( c ( d e ) f g h ( ( i ( j k ) ) l m n ) o p ) ( r s ) #

Contador del nivel de anidamiento:

Al encontrar '(' incrementamos – Al encontrar ')' decrementamos

Al terminar, el contador deberá tener el valor 0

Errores:

- Contador -1: paréntesis de cierre sin uno de apertura pendiente  
abc)de(fgh(ij))#
- Contador termina con un valor positivo  
Más paréntesis de apertura que de cierre  
Algún paréntesis sin cerrar: (a(b(cd(e)f)gh(i))jk#



# ¿Paréntesis bien emparejados?

Un error puede interrumpir el recorrido:

```
char c;  
int anidamiento = 0, pos = 0;  
bool error = false;  
cin >> c;  
while ((c != '#') && !error) {  
    pos++;  
    if (c == '(') {  
        anidamiento++;  
    }  
    else if (c == ')') {  
        anidamiento--;  
    }  
    if (anidamiento < 0) {  
        error = true;  
    }  
    if (!error) {  
        cin >> c;  
    }  
}
```

Luis Hernández Yáñez



# ¿Paréntesis bien emparejados?

parenthesis.cpp

```
if (error) {  
    cout << "Error: cierre sin apertura (pos. " << pos  
        << ")";  
}  
else if (anidamiento > 0) {  
    cout << "Error: Apertura sin cierre";  
}  
else {  
    cout << "Correcto";  
}  
cout << endl;
```

```
D:\Docencia\FP\2013-2014\Lessons\Less03\Examples\ESP>parenthesis  
ab(c(de) fgh((i(jk))lmn)op)(rs)#  
Correcto  
  
D:\Docencia\FP\2013-2014\Lessons\Less03\Examples\ESP>parenthesis  
)ab(((cd)ef))#  
Error: cierre sin apertura (pos. 1)  
  
D:\Docencia\FP\2013-2014\Lessons\Less03\Examples\ESP>parenthesis  
(((abc(d)e(fg(h))))#  
Error: Apertura sin cierre
```

Luis Hernández Yáñez



# ¿Dos secuencias iguales?

iguales.cpp

```
bool iguales() {
    bool sonIguales = true;
    double d1, d2;
    ifstream sec1, sec2;
    bool final = false;
    sec1.open("secuencia1.txt");
    sec2.open("secuencia2.txt");
    sec1 >> d1;
    sec2 >> d2; // Al menos estarán los centinelas (0)
    while (sonIguales && !final) {
        sonIguales = (d1 == d2);
        final = ((d1 == 0) || (d2 == 0));
        if (!final) {
            sec1 >> d1;
            sec2 >> d2;
        }
    }
    sec1.close();
    sec2.close();
    return sonIguales;
}
```



Cambia secuencia2.txt por secuencia3.txt y por secuencia4.txt para comprobar otros casos

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II (Anexo II)

Página 412



# Números primos menores que N

primos.cpp

Secuencia calculada: números divisibles sólo por 1 y ellos mismos ( $< N$ )

```
#include <iostream>
using namespace std;
bool primo(int n);
int main() {
    int num, candidato;
    cout << "Entero en el que parar (>1): ";
    cin >> num;
    if (num > 1) {
        candidato = 2; // El 1 no se considera un número primo
        while (candidato < num) {
            cout << candidato << " "; // Mostrar número primo
            candidato++;
            while (!primo(candidato)) { // Siguiendo primo
                candidato++;
            }
        }
    }
    return 0;
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II (Anexo II)

Página 413



# Números primos menores que N

```
bool primo(int n) {
    bool esPrimo = true;

    for (int i = 2; i <= n - 1; i++) {
        if (n % i == 0) {
            esPrimo = false; // Es divisible por i
        }
    }

    return esPrimo;
}
```



# Números primos menores que N

primos2.cpp

Mejoras: probar sólo impares; sólo pueden ser divisibles por impares; no pueden ser divisibles por ninguno mayor que su mitad

```
    candidato = 2;
    cout << candidato << " "; // Mostrar el número primo 2
    candidato++; // Seguimos con el 3, que es primo
    while (candidato < num) {
        cout << candidato << " "; // Mostrar número primo
        candidato = candidato + 2; // Sólo probamos impares
        while (!primo(candidato)) { // Siguiendo número primo
            candidato = candidato + 2;
        }
    } ...

bool primo(int n) {
    bool esPrimo = true;
    for (int i = 3; i <= n / 2; i = i + 2) {
        if (n % i == 0) {
            esPrimo = false; // Es divisible por i
        }
    }
    ...
}
```



Otra mejora más: Paramos al encontrar el primer divisor

```
bool primo(int n) {
    bool esPrimo = true;

    int i = 3;
    while ((i <= n / 2) && esPrimo) {
        if (n % i == 0) {
            esPrimo = false;
        }
        i = i + 2;
    }

    return esPrimo;
}
```



## Fundamentos de la programación

# Búsquedas





# Búsqueda de un número en un archivo

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

int busca(int n);
// Devuelve la línea en la que se encuentra o -1 si no está

int main() {
    int num, linea;

    cout << "Valor a localizar: ";
    cin >> num;
    linea = busca(num);
    if (linea != -1) {
        cout << "Encontrado (línea " << linea << ")" << endl;
    }
    else {
        cout << "No encontrado" << endl;
    }
    return 0;
}
```

buscaarch.cpp

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II (Anexo II)

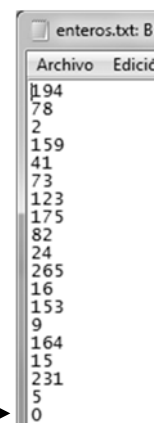
Página 418



# Búsqueda de un número en un archivo

```
int busca(int n) {
    int i, linea = 0;
    bool encontrado = false;
    ifstream archivo;
    archivo.open("enteros.txt");
    if (!archivo.is_open()) {
        linea = -1;
    }
    else {
        archivo >> i;
        while ((i != 0) && !encontrado) {
            linea++;
            if (i == n) {
                encontrado = true;
            }
            archivo >> i;
        }
        if (!encontrado) {
            linea = -1;
        }
        archivo.close();
    }
    return linea;
}
```

Centinela →



Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II (Anexo II)

Página 419



## Búsquedas en secuencias ordenadas

Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II (Anexo II)

Página 420



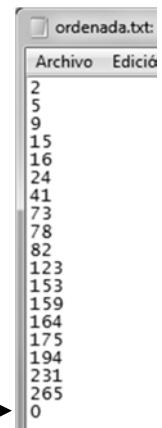
## Búsqueda en secuencias ordenadas

Secuencia ordenada de menor a mayor:  
paramos al encontrar uno mayor o igual al buscado

`buscaord.cpp`

Los que resten serán seguro mayores: *¡no puede estar el buscado!*

```
cout << "Valor a localizar: ";  
cin >> num;  
archivo >> i;  
while ((i != 0) && (i < num)) {  
    cont++;  
    archivo >> i;  
}  
if (i == num) {  
    cout << "Encontrado (pos.: " << cont << ")";  
}  
else {  
    cout << "No encontrado";  
}  
cout << endl;  
archivo.close();
```



Luis Hernández Yáñez



Fundamentos de la programación: Tipos e instrucciones II (Anexo II)

Página 421



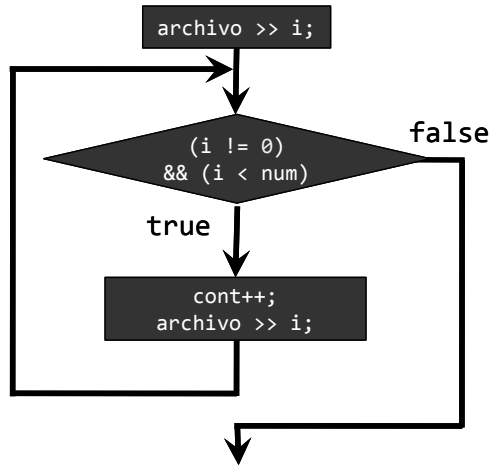
# Secuencias ordenadas

Si el elemento está: procesamiento similar a secuencias desordenadas

↓  
2 5 9 15 16 24 41 73 78 82 123 153 159 ...

num 9

i 9



Luis Hernández Yáñez



# Secuencias ordenadas

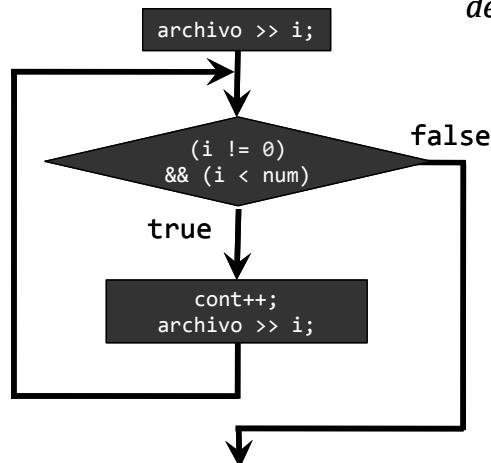
Si el elemento no está: evitamos buscar en el resto de la secuencia

↓  
2 5 9 15 16 24 41 73 78 82 123 153 159 ...

*No se procesa el resto de la secuencia*

num 10

i 15



Luis Hernández Yáñez








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.



## 4

# La abstracción procedimental

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Diseño descendente: Tareas y subtareas	427
Subprogramas	434
Subprogramas y datos	441
Parámetros	446
Argumentos	451
Resultado de la función	467
Prototipos	473
Ejemplos completos	475
Funciones de operador	477
Diseño descendente (un ejemplo)	480
Precondiciones y postcondiciones	490



## Diseño descendente Tareas y subtareas



## Tareas y subtareas

---

### *Refinamientos sucesivos*

Tareas que ha de realizar un programa:

Se pueden dividir en subtareas más sencillas

Subtareas:

También se pueden dividir en otras más sencillas...

→ Refinamientos sucesivos

Diseño en sucesivos pasos en los se amplía el detalle

Ejemplos:

- ✓ Dibujar 
- ✓ Mostrar la cadena HOLA MAMA en letras gigantes



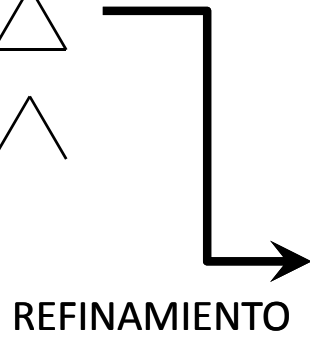
# Un dibujo



1. Dibujar ○

2. Dibujar △

3. Dibujar ^



1. Dibujar ○

2. Dibujar △

2.1. Dibujar ^

2.2. Dibujar —

3. Dibujar ^

Misma tarea



# Un dibujo



1. Dibujar ○

2. Dibujar △

2.1. Dibujar ^

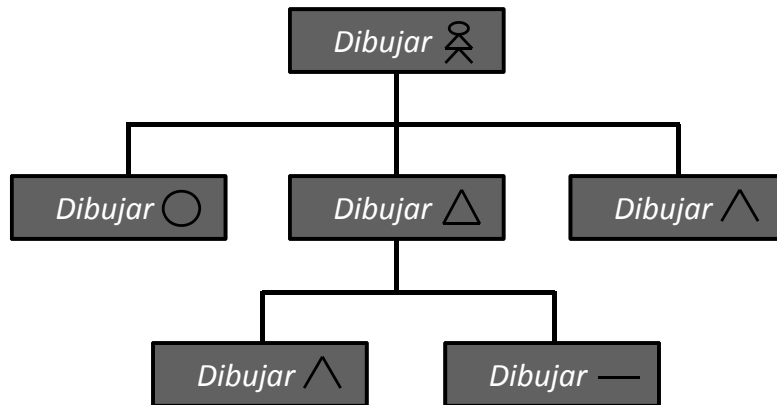
2.2. Dibujar —

3. Dibujar ^

4 tareas, pero dos de ellas son iguales  
Nos basta con saber cómo dibujar:



# Un dibujo



```
void dibujarCirculo()
{ ... }

void dibujarSecantes()
{ ... }

void dibujarLinea()
{ ... }

void dibujarTriangulo()
{
    dibujarSecantes();
    dibujarLinea();
}

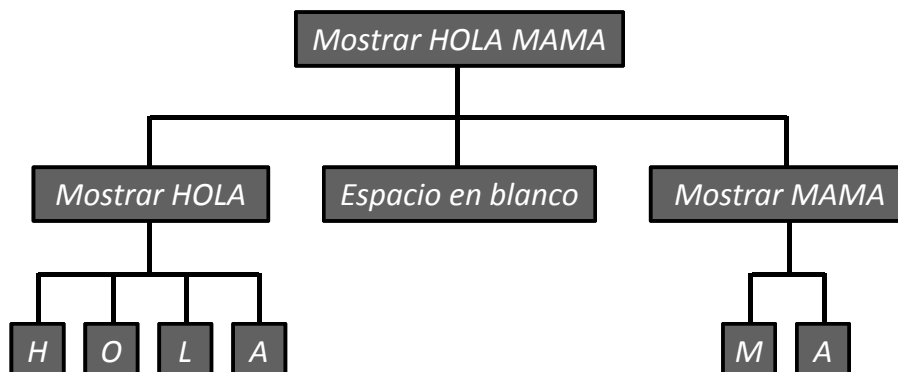
int main() {
    dibujarCirculo();
    dibujarTriangulo();
    dibujarSecantes();
    return 0;
}
```

Luis Hernández Yáñez



# Mensaje en letras gigantes

Mostrar la cadena HOLA MAMA en letras gigantes



Tareas básicas



Luis Hernández Yáñez





# Mensaje en letras gigantes

---

```
void mostrarH() {
    cout << "*   *" << endl;
    cout << "*   *" << endl;
    cout << "*****" << endl;
    cout << "*   *" << endl;
    cout << "*   *" << endl << endl;
}

void mostrarO() {
    cout << "*****" << endl;
    cout << "*   *" << endl;
    cout << "*   *" << endl;
    cout << "*   *" << endl;
    cout << "*****" << endl << endl;
}

void mostrarL()
{ ... }

void mostrarA()
{ ...}

void espaciosEnBlanco() {
    cout << endl << endl << endl;
}

void mostrarM()
{ ...}

int main() {
    mostrarH();
    mostrarO();
    mostrarL();
    mostrarA();
    espaciosEnBlanco();
    mostrarM();
    mostrarA();
    mostrarM();
    mostrarA();

    return 0;
}
```



# Fundamentos de la programación

---

## Subprogramas



# Abstracción procedimental

## Subprogramas

*Pequeños programas dentro de otros programas*

- ✓ Unidades de ejecución independientes
- ✓ Encapsulan código y datos
- ✓ Se comunican con otros subprogramas (datos)

*Subrutinas, procedimientos, funciones, acciones, ...*

- ✓ Realizan tareas individuales del programa
- ✓ Funcionalidad concreta, identificable y coherente (diseño)
- ✓ Se ejecutan de principio a fin cuando se llaman (*invocan*)
- ✓ Terminan devolviendo el control al punto de llamada

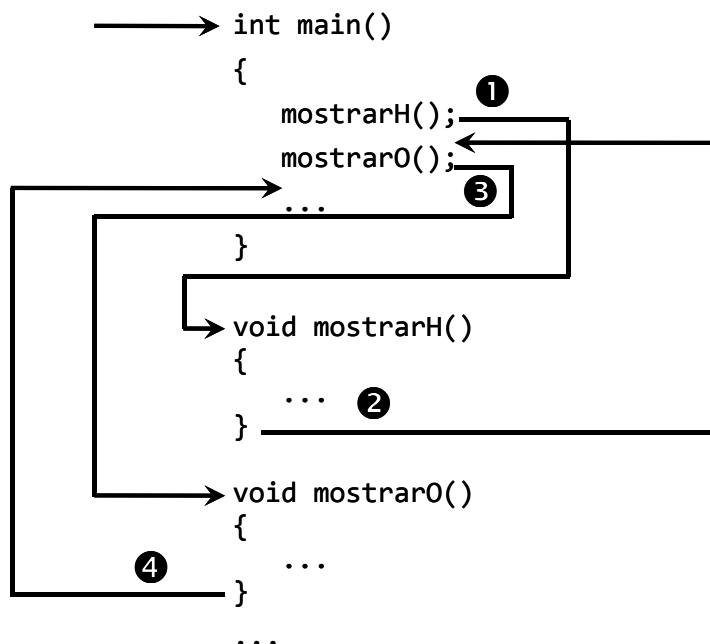


Aumentan el nivel de abstracción del programa  
Facilitan la prueba, la depuración y el mantenimiento



# Subprogramas

## Flujo de ejecución



# Subprogramas

---

## Subprogramas en C++

Forma general de un subprograma en C++:

```
tipo nombre(parámetros) // Cabecera
{
    // Cuerpo
}
```

- ✓ *Tipo* de dato que devuelve el subprograma como resultado
- ✓ *Parámetros* para la comunicación con el exterior
- ✓ *Cuerpo*: ¡Un bloque de código!



# Subprogramas

---

## Tipos de subprogramas

Procedimientos (*acciones*):

NO devuelven ningún resultado de su ejecución con `return`

Tipo: `void`

Llamada: instrucción independiente

`mostrarH()`;

Funciones:

SÍ devuelven un resultado con la instrucción `return`

Tipo distinto de `void`

Llamada: dentro de cualquier expresión

`x = 12 * y + cuadrado(20) - 3;`

Se sustituye en la expresión por el valor que devuelve

*¡Ya venimos utilizando funciones desde el Tema 2!*



# Subprogramas

## Funciones

### Subprogramas de tipo distinto de void

```
...
int menu()
{
  int op;
  cout << "1 - Editar" << endl;
  cout << "2 - Combinar" << endl;
  cout << "3 - Publicar" << endl;
  cout << "0 - Cancelar" << endl;
  cout << "Elija: ";
  cin >> op;
  return op;
}

int main()
{
  ...
  int opcion;
  opcion = menu();
  ...
}
```

Luis Hernández Yáñez



# Subprogramas

## Procedimientos

### Subprogramas de tipo void

```
...
void menu()
{
  int op;
  cout << "1 - Editar" << endl;
  cout << "2 - Combinar" << endl;
  cout << "0 - Cancelar" << endl;
  cout << "Opción: ";
  cin >> op;
  if (op == 1) {
    editar();
  }
  else if (op == 2) {
    combinar();
  }
}

int main()
{
  ...
  menu();
  ...
}
```

Luis Hernández Yáñez



## Subprogramas y datos



## Datos en los subprogramas

---

### *De uso exclusivo del subprograma*

```
tipo nombre(parámetros) // Cabecera
{
    Declaraciones Locales // Cuerpo
}
```

- ✓ Declaraciones locales de tipos, constantes y variables  
Dentro del cuerpo del subprograma
- ✓ Parámetros declarados en la cabecera del subprograma  
Comunicación del subprograma con otros subprogramas



# Datos locales y datos globales

## Datos en los programas

- ✓ Datos globales: declarados fuera de todos los subprogramas  
Existen durante toda la ejecución del programa
- ✓ Datos locales: declarados en algún subprograma  
Existen sólo durante la ejecución del subprograma

## Ámbito y visibilidad de los datos

Tema 3

- Ámbito de los datos globales: resto del programa  
Se conocen dentro de los subprogramas que siguen
- Ámbito de los datos locales: resto del subprograma  
No se conocen fuera del subprograma
- Visibilidad de los datos  
Datos locales a un bloque ocultan otros externos homónimos



# Datos locales y datos globales

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

```
const int MAX = 100;
double ingresos;
```

} Datos globales

```
...
void proc() {
    int op;
    double ingresos;
```

} Datos locales a proc()

... → Se conocen MAX (global), op (local)  
e ingresos (local que oculta la global)

```
int main() {
    int op;
    ...
    return 0;
}
```

} Datos locales a main()

... → Se conocen MAX (global), op (local)  
e ingresos (global)

op de proc() es distinta de op de main()



# Datos locales y datos globales

---

## *Sobre el uso de datos globales en los subprogramas*

NO SE DEBEN USAR datos globales en subprogramas

✓ *¿Necesidad de datos externos?*

Define parámetros en el subprograma

Los datos externos se pasan como argumentos en la llamada

✓ Uso de datos globales en los subprogramas:

Riesgo de *efectos laterales*

Modificación inadvertida de esos datos afectando otros sitios

Excepciones:

✓ Constantes globales (valores inalterables)

✓ Tipos globales (necesarios en varios subprogramas)



# Fundamentos de la programación

---

## Parámetros

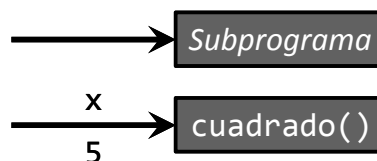


# Comunicación con el exterior

Datos de entrada, datos de salida y datos de entrada/salida

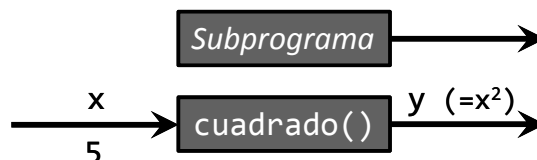
Datos de entrada: Aceptados

Subprograma que dado un número muestra en la pantalla su cuadrado:



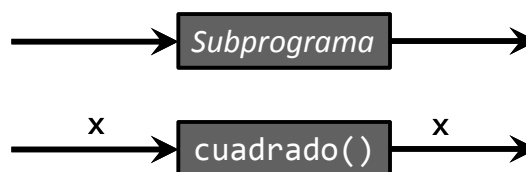
Datos de salida: Devueltos

Subprograma que dado un número devuelve su cuadrado:



Datos de entrada/salida: Aceptados y modificados

Subprograma que dada una variable numérica la eleva al cuadrado:



Luis Hernández Yáñez



# Parámetros en C++

## Declaración de parámetros

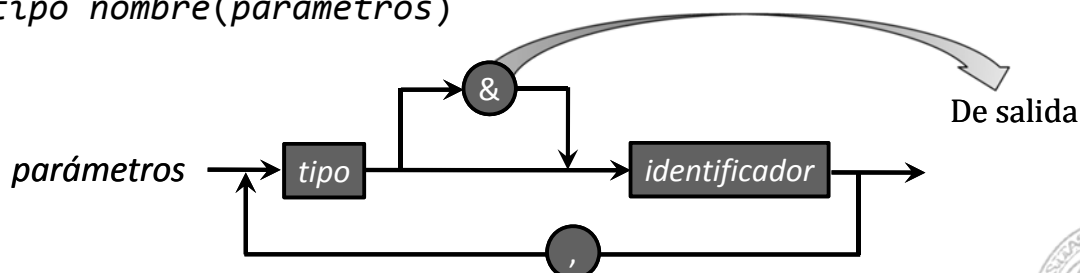
Sólo dos clases de parámetros en C++:

- Sólo de entrada (*por valor*)
- De salida (sólo salida o E/S) (*por referencia / por variable*)

## Lista de parámetros formales

Entre los paréntesis de la cabecera del subprograma

*tipo nombre(parámetros)*



Luis Hernández Yáñez





# Parámetros por valor

---

Reciben copias de los argumentos usados en la llamada

```
int cuadrado(int num)
```

```
double potencia(double base, int exp)
```

```
void muestra(string nombre, int edad, string nif)
```

```
void proc(char c, int x, double a, bool b)
```

Reciben sus valores en la llamada del subprograma

Argumentos: Expresiones en general

Variables, constantes, literales, llamadas a función, operaciones

Se destruyen al terminar la ejecución del subprograma

*¡Atención!* Los arrays se pasan por valor como constantes:

```
double media(const TArray lista)
```



# Parámetros por referencia

---



Misma identidad que la variable pasada como argumento

```
void incrementa(int &x)
```

```
void intercambia(double &x, double &y)
```

```
void proc(char &c, int &x, double &a, bool &b)
```

Reciben las variables en la llamada del subprograma: *¡Variables!*

Los argumentos pueden quedar modificados

*¡No usaremos parámetros por valor en las funciones!*

Sólo en procedimientos



Puede haber tanto por valor como por referencia

*¡Atención!* Los arrays se pasan por referencia sin utilizar &

```
void insertar(TArray lista, int &contador, double item)
```

El argumento de lista (variable TArray) quedará modificado



## Argumentos



## Llamada a subprogramas con parámetros

---

*nombre(argumentos)*

- Tantos argumentos como parámetros y en el mismo orden
- Concordancia de tipos argumento-parámetro
- Por valor: Expresiones válidas (se pasa el resultado)
- Por referencia: ¡*Sólo variables!*

Se copian los valores de las expresiones pasadas por valor  
en los correspondientes parámetros

Se hacen corresponder los argumentos pasados por referencia  
(variables) con sus correspondientes parámetros



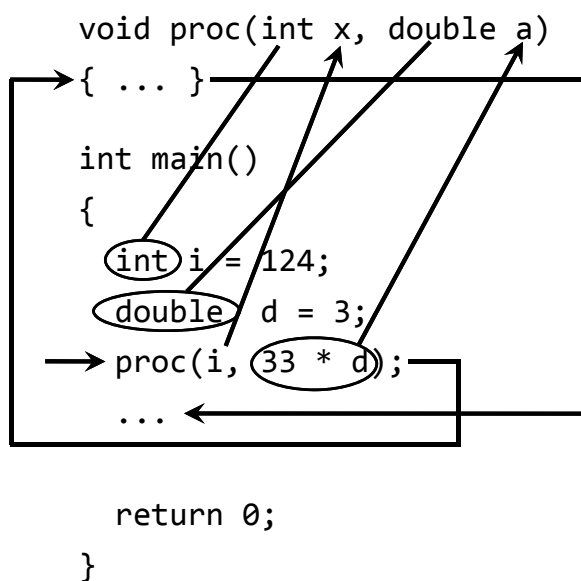
# Argumentos pasados por valor

Expresiones válidas con concordancia de tipo:

```
void proc(int x, double a) → proc(23 * 4 / 7, 13.5);  
→ double d = 3;  
   proc(12, d);  
→ double d = 3;  
   int i = 124;  
   proc(i, 33 * d);  
→ double d = 3;  
   int i = 124;  
   proc(cuad(20) * 34 + i, i * d);
```



# Argumentos pasados por valor



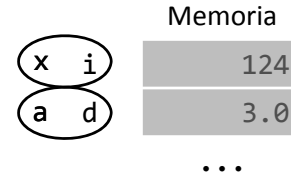
	Memoria
i	124
d	3.0
	...
	...
x	124
a	99.0
	...



# Argumentos pasados por referencia

```
void proc(int &x, double &a)
{ ... }

int main()
{
  int i = 124;
  double d = 3;
  proc(i, d);
  ...
  return 0;
}
```



## ¿Qué llamadas son correctas?

Dadas las siguientes declaraciones:

```
int i;
double d;
void proc(int x, double &a);
```

¿Qué pasos de argumentos son correctos? ¿Por qué no?

- proc(3, i, d);            **✗**    N° de argumentos ≠ N° de parámetros
- proc(i, d);                **✓**
- proc(3 \* i + 12, d);    **✓**
- proc(i, 23);              **✗**    Parámetro por referencia → ¡variable!
- proc(d, i);                **✗**    ¡Argumento double para parámetro int!
- proc(3.5, d);            **✗**    ¡Argumento double para parámetro int!
- proc(i);                  **✗**    N° de argumentos ≠ N° de parámetros



# Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            divide(i, j, cociente, resto);
            cout << i << " entre " << j << " da un cociente de "
                << cociente << " y un resto de " << resto << endl;
        }
    }

    return 0;
}
```

Luis Hernández Yáñez



# Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            → divide(i, j, cociente, resto);
            ...
        }
    }

    return 0;
}
```

	Memoria
cociente	?
resto	?
i	1
j	1
...	...

Luis Hernández Yáñez



# Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            divide(i, j, cociente, resto);
        }
    }

    return 0;
}
```

		Memoria
div	cociente	?
rem	resto	?
i		1
j		1
		...
op1		1
op2		1
		...

Luis Hernández Yáñez



# Paso de argumentos

```
...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            divide(i, j, cociente, resto);
        }
    }

    return 0;
}
```

		Memoria
div	cociente	1
rem	resto	0
i		1
j		1
		...
op1		1
op2		1
		...

Luis Hernández Yáñez



# Paso de argumentos

```

...
void divide(int op1, int op2, int &div, int &rem) {
// Divide op1 entre op2 y devuelve el cociente y el resto
    div = op1 / op2;
    rem = op1 % op2;
}

int main() {
    int cociente, resto;
    for (int j = 1; j <= 4; j++) {
        for (int i = 1; i <= 4; i++) {
            divide(i, j, cociente, resto);
            ...
        }
    }
    return 0;
}

```

Memoria	
cociente	1
resto	0
i	1
j	1
...	

Luis Hernández Yáñez



# Más ejemplos

```

...
void intercambia(double &valor1, double &valor2) {
// Intercambia los valores
→ double tmp; // Variable local (temporal)
    tmp = valor1;
    valor1 = valor2;
    valor2 = tmp;
}

int main() {
    double num1, num2;
    cout << "Valor 1: ";
    cin >> num1;
    cout << "Valor 2: ";
    cin >> num2;
    intercambia(num1, num2);
    cout << "Ahora el valor 1 es " << num1
        << " y el valor 2 es " << num2 << endl;
    return 0;
}

```

Memoria temporal del procedimiento	
tmp	?
...	

Memoria de main()		
valor1	num1	13.6
valor2	num2	317.14
...		

Luis Hernández Yáñez



## Más ejemplos

---

```
...
// Prototipo
void cambio(double precio, double pago, int &euros, int &cent50,
            int &cent20, int &cent10, int &cent5, int &cent2, int &cent1);

int main() {
    double precio, pago;
    int euros, cent50, cent20, cent10, cent5, cent2, cent1;
    cout << "Precio: ";
    cin >> precio;
    cout << "Pago: ";
    cin >> pago;
    cambio(precio, pago, euros, cent50, cent20, cent10, cent5, cent2,
           cent1);
    cout << "Cambio: " << euros << " euros, " << cent50 << " x 50c., "
         << cent20 << " x 20c., " << cent10 << " x 10c., "
         << cent5 << " x 5c., " << cent2 << " x 2c. y "
         << cent1 << " x 1c." << endl;

    return 0;
}
```

Luis Hernández Yáñez



## Más ejemplos

---

```
void cambio(double precio, double pago, int &euros, int &cent50,
            int &cent20, int &cent10, int &cent5, int &cent2, int &cent1) {
    if (pago < precio) { // Cantidad insuficiente
        cout << "Error: El pago es inferior al precio" << endl;
    }
    else {
        int cantidad = int(100.0 * (pago - precio) + 0.5);
        euros = cantidad / 100;
        cantidad = cambio % 100;
        cent50 = cantidad / 50;
        cantidad = cantidad % 50;
        cent20 = cantidad / 20;
        cantidad = cantidad % 20;
        cent10 = cantidad / 10;
        cantidad = cantidad % 10;
        cent5 = cantidad / 5;
        cantidad = cantidad % 5;
        cent2 = cantidad / 2;
        cent1 = cantidad % 2;
    }
}
```



Explicación en el libro de  
Adams/Leestma/Nyhoff

Luis Hernández Yáñez





# Notificación de errores

En los subprogramas se pueden detectar errores

Errores que impiden realizar los cálculos:

```
void cambio(double precio, double pago, int &euros, int &cent50,
            int &cent20, int &cent10, int &cent5, int &cent2, int &cent1) {
    → {
        if (pago < precio) { // Cantidad insuficiente
            cout << "Error: El pago es inferior al precio" << endl;
        }
        ...
    }
```

¿Debe el subprograma notificar al usuario o al programa?

→ Mejor notificarlo al punto de llamada y allí decidir qué hacer

```
void cambio(double precio, double pago, int &euros, int &cent50,
            int &cent20, int &cent10, int &cent5, int &cent2, int &cent1,
            → bool &error) {
    if (pago < precio) { // Cantidad insuficiente
        → error = true;
    }
    else {
        → error = false;
    }
    ...
}
```

Luis Hernández Yáñez



# Notificación de errores

cambio.cpp

Al volver de la llamada se decide qué hacer si ha habido error...

- ✓ ¿Informar al usuario?
- ✓ ¿Volver a pedir los datos?
- ✓ Etcétera

```
int main() {
    double precio, pago;
    int euros, cent50, cent20, cent10, cent5, cent2, cent1;
    → bool error;
    cout << "Precio: ";
    cin >> precio;
    cout << "Pago: ";
    cin >> pago;
    cambio(precio, pago, euros, cent50, cent20, cent10, cent5, cent2,
           cent1, error);
    → if (error) {
        cout << "Error: El pago es inferior al precio" << endl;
    }
    else {
        ...
    }
}
```

Luis Hernández Yáñez



## Resultado de la función



## Resultado de la función

*Una función ha de devolver un resultado*

La función ha de terminar su ejecución devolviendo el resultado

La instrucción `return`:

- Devuelve el dato que se indica a continuación como resultado
- Termina la ejecución de la función

El dato devuelto sustituye a la llamada de la función en la expresión

```
int cuad(int x) {  
    return x * x;  
    x = x * x;  
}  
  
int main() {  
    cout << 2 * cuad(16);  
    return 0;  
}
```

Diagrama de flujo: Una flecha apunta desde la llamada `cuad(16)` en `main` hasta la instrucción `return x * x;` en `cuad`. Otra flecha apunta desde `return x * x;` hasta el valor `256` que aparece a la derecha. Una tercera flecha apunta desde `return 0;` en `main` hacia abajo y a la izquierda, con el texto "Esta instrucción no se ejecutará nunca" debajo de ella.



# Ejemplo: Cálculo del factorial

factorial.cpp

Factorial (N) = 1 x 2 x 3 x ... x (N-2) x (N-1) x N

```
long long int factorial(int n); // Prototipo

int main() {
    int num;
    cout << "Num: ";
    cin >> num;
    cout << "Factorial de " << num << ": " << factorial(num) << endl;
    return 0;
}

long long int factorial(int n) {
    long long int fact = 1;
    if (n < 0) {
        fact = 0;
    }
    else {
        for (int i = 1; i <= n; i++) {
            fact = fact * i;
        }
    }
    → return fact;
}
```

Luis Hernández Yáñez



Fundamentos de la programación: La abstracción procedimental

Página 469



## Un único punto de salida

```
int compara(int val1, int val2) {
    // -1 si val1 < val2, 0 si iguales, +1 si val1 > val2
    if (val1 == val2) {
        return 0;
    }
    else if (val1 < val2) {
        return -1;
    }
    else {
        return 1;
    }
}
```

¡3 puntos de salida!



Luis Hernández Yáñez



Fundamentos de la programación: La abstracción procedimental

Página 470



# Un único punto de salida

```
int compara(int val1, int val2) {  
  // -1 si val1 < val2, 0 si iguales, +1 si val1 > val2  
  int resultado;  
  
  if (val1 == val2) {  
    resultado = 0;  
  }  
  else if (val1 < val2) {  
    resultado = -1;  
  }  
  else {  
    resultado = 1;  
  }  
  
  return resultado; }  
}
```

→ Punto de salida único



# ¿Cuándo termina el subprograma?

Procedimientos (tipo void):

- Al encontrar la llave de cierre que termina el subprograma
- Al encontrar una instrucción `return` (sin resultado)

Funciones (tipo distinto de void):

- SÓLO al encontrar una instrucción `return` (con resultado)

Nuestros subprogramas siempre terminarán al final:

- ✓ No usaremos `return` en los procedimientos
- ✓ Funciones: sólo un `return` y estará al final



Para facilitar la depuración y el mantenimiento, codifica los subprogramas con un único punto de salida



## Prototipos



## ¿Qué subprogramas hay en el programa?

---

¿Dónde los ponemos? ¿Antes de `main()`? ¿Después de `main()`?

→ Los pondremos después de `main()`

¿Son correctas las llamadas a subprogramas?

En `main()` o en otros subprogramas

– ¿Existe el subprograma?

– ¿Concuerdan los argumentos con los parámetros?

Deben estar los prototipos de los subprogramas antes de `main()`

Prototipo: cabecera del subprograma terminada en ;

```
void dibujarCirculo();  
void mostrarM();  
void proc(double &a);  
int cuad(int x);  
...
```



`main()` es el único subprograma que no hay que prototipar



# Ejemplos

intercambia.cpp

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

void intercambia(double &valor1, double &valor2); // Prototipo

int main() {
    double num1, num2;
    cout << "Valor 1: ";
    cin >> num1;
    cout << "Valor 2: ";
    cin >> num2;
    intercambia(num1, num2);
    cout << "Ahora el valor 1 es " << num1
         << " y el valor 2 es " << num2 << endl;
    return 0;
}

void intercambia(double &valor1, double &valor2) {
    double tmp; // Variable local (temporal)
    tmp = valor1;
    valor1 = valor2;
    valor2 = tmp;
}
```



Asegúrate de que los prototipos coincidan con las implementaciones

Luis Hernández Yáñez



Fundamentos de la programación: La abstracción procedimental

Página 475



# Ejemplos

mates.cpp

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

// Prototipos
long long int factorial(int n);
int sumatorio(int n);

int main() {
    int num;
    cout << "Num: ";
    cin >> num;
    cout << "Factorial de "
         << num << ": "
         << factorial(num)
         << endl
         << "Sumatorio de 1 a "
         << num << ": "
         << sumatorio(num)
         << endl;

    return 0;
}

long long int factorial(int n) {
    long long int fact = 1;

    if (n < 0) {
        fact = 0;
    }
    else {
        for (int i = 1; i <= n; i++) {
            fact = fact * i;
        }
    }

    return fact;
}

int sumatorio(int n) {
    int sum = 0;

    for (int i = 1; i <= n; i++) {
        sum = sum + i;
    }

    return sum;
}
```

Luis Hernández Yáñez



Fundamentos de la programación: La abstracción procedimental

Página 476



## Funciones de operador



## Funciones de operador

---

### *Notación infija (de operador)*

*operandoIzquierdo operador operandoDerecho*

$a + b$

Se ejecuta el operador con los operandos como argumentos

Los operadores se implementan como funciones:

*tipo operatorsímbolo(parámetros)*

Si es un operador monario sólo habrá un parámetro

Si es binario habrá dos parámetros

El *símbolo* es un símbolo de operador (uno o dos caracteres):

$+, -, *, /, --, <<, \%, \dots$



# Funciones de operador

---

```
tMatriz suma(tMatriz a, tMatriz b);
```

```
tMatriz a, b, c;  
c = suma(a, b);
```

```
tMatriz operator+(tMatriz a, tMatriz b);
```

```
tMatriz a, b, c;  
c = a + b;
```

¡La implementación será exactamente la misma!

Mayor aproximación al lenguaje matemático



# Fundamentos de la programación

---

## Diseño descendente (un ejemplo)





# Refinamientos sucesivos

---

Especificación inicial (Paso 0).-

*Desarrollar un programa que haga operaciones de conversión de medidas hasta que el usuario decida que no quiere hacer más*

Análisis y diseño aumentando el nivel de detalle en cada paso

*¿Qué operaciones de conversión?*

Paso 1.-

*Desarrollar un programa que haga operaciones de conversión de medidas hasta que el usuario decida que no quiere hacer más*

- \* *Pulgadas a centímetros*
- \* *Libras a gramos*
- \* *Grados Fahrenheit a centígrados*
- \* *Galones a litros*



# Refinamientos sucesivos

---

Paso 2.-

*Desarrollar un programa que muestre al usuario un menú con cuatro operaciones de conversión de medidas:*

- \* *Pulgadas a centímetros*
- \* *Libras a gramos*
- \* *Grados Fahrenheit a centígrados*
- \* *Galones a litros*

*Y lea la elección del usuario y proceda con la conversión, hasta que el usuario decida que no quiere hacer más*

6 grandes tareas:

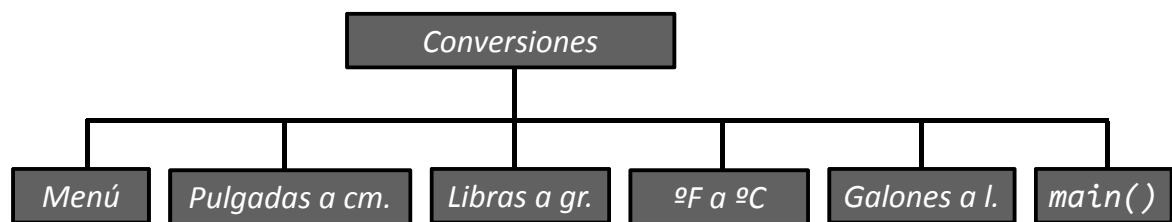
Menú, cuatro funciones de conversión y `main()`



# Refinamientos sucesivos

---

Paso 2.-



# Refinamientos sucesivos

---

Paso 3.-

★ *Menú:*

Mostrar las cuatro opciones más una para salir  
Validar la entrada y devolver la elegida

★ *Pulgadas a centímetros:*

Devolver el equivalente en centímetros del valor en pulgadas

★ *Libras a gramos:*

Devolver el equivalente en gramos del valor en libras

★ *Grados Fahrenheit a centígrados:*

Devolver el equivalente en centígrados del valor en Fahrenheit

★ *Galones a litros:*

Devolver el equivalente en litros del valor en galones

★ *Programa principal (main())*



# Refinamientos sucesivos

---

Paso 3.- Cada tarea, un subprograma

Comunicación entre los subprogramas:

Función	Entrada	Salida	Valor devuelto
menu()	—	—	int
pulgACm()	double pulg	—	double
lbAGr()	double libras	—	double
grFAGrC()	double grF	—	double
galALtr()	double galones	—	double
main()	—	—	int



# Refinamientos sucesivos

---

Paso 4.- Algoritmos detallados de cada subprograma → Programar

```
#include <iostream>
using namespace std;
// Prototipos
int menu();
double pulgACm(double pulg);
double lbAGr(double libras);
double grFAGrC(double grF);
double galALtr(double galones);
```

```
int main() {
    double valor;
    int op = -1;
    while (op != 0) {
        op = menu();
        switch (op) {
            case 1:
                {
                    cout << "Pulgadas: ";
                    cin >> valor;
                    cout << "Son " << pulgACm(valor) << " cm." << endl;
                }
                break;
                ...
            }
    }
}
```



# Refinamientos sucesivos

---

```
case 2:
{
    cout << "Libras: ";
    cin >> valor;
    cout << "Son " << lbAGr(valor) << " gr." << endl;
}
break;
case 3:
{
    cout << "Grados Fahrenheit: ";
    cin >> valor;
    cout << "Son " << grFAGrC(valor) << " °C" << endl;
}
break;
case 4:
{
    cout << "Galones: ";
    cin >> valor;
    cout << "Son " << galALtr(valor) << " l." << endl;
}
break;
}
return 0;
}
```

Luis Hernández Yáñez



# Refinamientos sucesivos

---

```
int menu() {
    int op = -1;

    while ((op < 0) || (op > 4)) {
        cout << "1 - Pulgadas a Cm." << endl;
        cout << "2 - Libras a Gr." << endl;
        cout << "3 - Fahrenheit a °C" << endl;
        cout << "4 - Galones a L." << endl;
        cout << "0 - Salir" << endl;
        cout << "Elige: ";
        cin >> op;
        if ((op < 0) || (op > 4)) {
            cout << "Opción no válida" << endl;
        }
    }

    return op;
}

double pulgACm(double pulg) {
    const double cmPorPulg = 2.54;
    return pulg * cmPorPulg;
}
```

Luis Hernández Yáñez



```
double lbAGr(double libras) {  
    const double grPorLb = 453.6;  
    return libras * grPorLb;  
}  
  
double grFAGrC(double grF) {  
    return ((grF - 32) * 5 / 9);  
}  
  
double galALtr(double galones) {  
    const double ltrPorGal = 4.54609;  
    return galones * ltrPorGal;  
}
```



## Fundamentos de la programación

# Precondiciones y postcondiciones



# Precondiciones y postcondiciones

---

## *Integridad de los subprogramas*

Condiciones que se deben dar antes de comenzar su ejecución

→ Precondiciones

✓ Quien llame al subprograma debe garantizar que se satisfacen

Condiciones que se darán cuando termine su ejecución

→ Postcondiciones

✓ En el punto de llamada se pueden dar por garantizadas

*Aserciones:*

Condiciones que si no se cumplen interrumpen la ejecución

Función `assert()`



# Aserciones como precondiciones

---

## *Precondiciones*

Por ejemplo, no realizaremos conversiones de valores negativos:

```
double pulgACm(double pulg) {
```

```
    assert(pulg > 0);
```

```
    double cmPorPulg = 2.54;
```

```
    return pulg * cmPorPulg;
```

```
}
```

La función tiene una precondición: `pulg` debe ser positivo

`assert(pulg > 0);` interrumpirá la ejecución si no es cierto



# Aserciones como precondiciones

## Precondiciones

Es responsabilidad del punto de llamada garantizar la precondición:

```
int main() {
    double valor;
    int op = -1;
    while (op != 0) {
        op = menu();
        switch (op) {
            case 1:
                {
                    cout << "Pulgadas: ";
                    cin >> valor;
                    if (valor < 0) {
                        cout << "¡No válido!" << endl;
                    }
                    else { // Se cumple la precondición...
                        ...
                    }
                }
        }
    }
}
```

Luis Hernández Yáñez



# Aserciones como postcondiciones

## Postcondiciones

Un subprograma puede garantizar condiciones al terminar:

```
int menu() {
    int op = -1;
    while ((op < 0) || (op > 4)) {
        ...
        cout << "Elige: ";
        cin >> op;
        if ((op < 0) || (op > 4)) {
            cout << "Opción no válida" << endl;
        }
    }
    assert ((op >= 0) && (op <= 4));
    return op;
}
```

Luis Hernández Yáñez






El subprograma debe asegurarse de que se cumpla



## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.







ANEXO

## Más sobre subprogramas

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Archivos como parámetros	498
La función <code>main()</code>	501
Argumentos implícitos	504
Sobrecarga de subprogramas	508



## Archivos como parámetros



## Archivos como parámetros

---

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

void sumatorio_archivo(ifstream &arch, double &suma);

int main() {
    double resultado;
    ifstream archivo;
    archivo.open("datos.txt");
    if (!archivo.is_open()) {
        cout << "ERROR DE APERTURA" << endl;
    }
    else {
        sumatorio_archivo(archivo, resultado)
        cout << "Suma = " << resultado << endl;
        archivo.close();
    }

    return 0;
}
```



# Archivos como parámetros

---

```
void sumatorio_archivo(ifstream &arch, double &suma) {  
    double dato;  
  
    suma = 0;  
    arch >> dato;  
  
    while (dato != -1) {  
        suma = suma + dato;  
        arch >> dato;  
    }  
}
```



Los archivos siempre se pasan por referencia



# Fundamentos de la programación

---

## La función main()



# Parámetros de `main()`

---

## *Comunicación con el sistema operativo*

Parámetros opcionales de la función `main()`:

```
int main(int argc, char *argv[])
```

Para obtener datos proporcionados al ejecutar el programa:

```
C:\>prueba cad1 cad2 cad3
```

Ejecuta `prueba.exe` con tres argumentos (cadenas)

Parámetros de `main()`:

- `argc`: número de argumentos que se proporcionan  
4 en el ejemplo (primero: nombre del programa con su ruta)
- `argv`: array con las cadenas proporcionadas como argumentos



# Lo que devuelve `main()`

---

## *¿Cómo ha ido la función?*

La función `main()` devuelve al S.O. un código de terminación

- `0`: *Todo OK*
- Distinto de `0`: *¡Ha habido un error!*

Si la ejecución llega al final de la función `main()`, todo OK:

```
...  
return 0; // Fin del programa  
}
```



## Argumentos implícitos



## Argumentos implícitos

---

### *Valores predeterminados para parámetros por valor*

Valor por defecto para un parámetro:

Tras un = a continuación del nombre del parámetro:

```
void proc(int i = 1);
```

Si no se proporciona argumento, el parámetro toma ese valor

```
proc(12);      i toma el valor explícito 12
```

```
proc();       i toma el valor implícito (1)
```

Sólo puede haber argumentos implícitos en los parámetros finales:

```
void p(int i, int j = 2, int k = 3); // CORRECTO
```

```
void p(int i = 1, int j, int k = 3); // INCORRECTO
```



Una vez asignado un valor implícito, todos los que siguen han de tener también valor implícito



# Argumentos implícitos

## Parámetros y argumentos implícitos

```
void p(int i, int j = 2, int k = 3);
```

Se copian los argumentos en los parámetros del primero al último  
→ los que no tengan correspondencia tomarán los implícitos

```
void p(int i, int j = 2, int k = 3);
```

...

```
p(13); // i toma 13, j y k sus valores implícitos
```

```
p(5, 7); // i toma 5, j toma 7 y k su valor implícito
```

```
p(3, 9, 12); // i toma 3, j toma 9 y k toma 12
```



Los argumentos implícitos se declaran en el prototipo (preferible) o en la cabecera del subprograma, pero NO en ambos



## Ejemplo

Por defecto, signo +  
Por defecto,  $\Delta$  es 1

$$f(x, y) = \pm \Delta \frac{x}{y}$$

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

```
double f(double x, double y, int signo = 1, double delta = 1.0);
```

```
int main() {  
    double x, y;  
    cout << "X = ";  
    cin >> x;  
    cout << "Y = ";  
    cin >> y;  
    cout << "signo y delta por defecto: " << f(x, y) << endl;  
    cout << "signo -1 y delta por defecto: " << f(x, y, -1) << endl;  
    cout << "signo y delta concretos: " << f(x, y, 1, 1.25) << endl;
```

```
    return 0;  
}
```

```
double f(double x, double y, int signo, double delta) {  
    return signo * delta * x / y;  
}
```



No podemos dejar signo por defecto y concretar delta



## Sobrecarga de subprogramas



## Sobrecarga de subprogramas

---

### *Igual nombre, distintos parámetros*

Funciones o procedimientos con igual nombre y distintos parámetros:

```
int abs(int n);  
double abs(double n);  
long int abs(long int n);
```

Se ejecutará la función que corresponda al tipo de argumento:

```
abs(13)    // argumento int --> primera función  
abs(-2.3) // argumento double --> segunda función  
abs(3L)    // argumento long int --> tercera función
```



Para indicar que es un literal `long int`, en lugar de `int`



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

void intercambia(int &x, int &y);
void intercambia(double &x,
                  double &y);
void intercambia(char &x, char &y);

void intercambia(int &x, int &y) {
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}

void intercambia(double &x,
                  double &y) {
    double tmp;
    tmp = x;
    x = y;
    y = tmp;
}

void intercambia(char &x, char &y) {
    char tmp;
    tmp = x;
    x = y;
    y = tmp;
}

int main() {
    int i1 = 3, i2 = 7;
    double d1 = 12.5, d2 = 35.9;
    char c1 = 'a', c2 = 'b';
    cout << i1 << " - " << i2 << endl;
    cout << d1 << " - " << d2 << endl;
    cout << c1 << " - " << c2 << endl;
    intercambia(i1, i2);
    intercambia(d1, d2);
    intercambia(c1, c2);
    cout << i1 << " - " << i2 << endl;
    cout << d1 << " - " << d2 << endl;
    cout << c1 << " - " << c2 << endl;
    return 0;
}
```

Luis Hernández Yáñez






## Acerca de *Creative Commons*



### *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.

Luis Hernández Yáñez





## 5

# Tipos de datos estructurados

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez/Pablo Moreno Ger  
Facultad de Informática  
Universidad Complutense



## Índice

---

Tipos de datos	514
Arrays de nuevo	517
Arrays y bucles for	520
Más sobre arrays	522
Inicialización de arrays	523
Enumerados como índices	524
Paso de arrays a subprogramas	525
Implementación de listas	528
Cadenas de caracteres	531
Cadenas de caracteres de tipo string	535
Entrada/salida con string	539
Operaciones con string	541
Estructuras	543
Estructuras dentro de estructuras	549
Arrays de estructuras	550
Arrays dentro de estructuras	551
Listas de longitud variable	552
Un ejemplo completo	558
El bucle do..while	562



## Tipos de datos



## Tipos de datos

---

### *Clasificación de tipos*

- ✓ **Simple**
  - ❖ Estándar: `int`, `float`, `double`, `char`, `bool`  
Conjunto de valores predeterminado ✓
  - ❖ Definidos por el usuario: *enumerados*  
Conjunto de valores definido por el programador ✓
  
- ✓ **Estructurados**
  - ❖ Colecciones homogéneas: *arrays*  
Todos los elementos del mismo tipo ✓
  - ❖ Colecciones heterogéneas: *estructuras*  
Los elementos pueden ser de tipos distintos



# Tipos estructurados

---

## *Colecciones o tipos aglomerados*

Agrupaciones de datos (elementos):

- ✓ Todos del mismo tipo: *array* o *tabla*
- ✓ De tipos distintos: *estructura*, *registro* o *tupla*

Arrays (tablas)

- Elementos organizados por posición: 0, 1, 2, 3, ...
- Acceso por índice: 0, 1, 2, 3, ...
- Una o varias dimensiones

Estructuras (tuplas, registros)

- Elementos (campos) sin orden establecido
- Acceso por nombre



# Fundamentos de la programación

---

## Arrays de nuevo



# Arrays

## Estructura secuencial

Cada elemento se encuentra en una posición (*índice*):

- ✓ Los índices son enteros positivos
- ✓ El índice del primer elemento siempre es 0
- ✓ Los índices se incrementan de uno en uno

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
	0	1	2	3	4	5	6

### Acceso directo

A cada elemento se accede a través de su índice:

`ventas[4]` accede al 5º elemento (contiene el valor 435.00)

```
cout << ventas[4];
```

```
ventas[4] = 442.75;
```



Datos de un mismo tipo base:  
Se usan como cualquier variable

[ ]



# Tipos y variables arrays

## Declaración de tipos de arrays

```
const int Dimensión = ...;
```

```
typedef tipo_base tNombre[Dimensión];
```

Ejemplo:

```
const int Dias = 7;
```

```
typedef double tVentas[Dias];
```

Declaración de variables de tipos array: como cualquier otra

```
tVentas ventas;
```

¡NO se inicializan los elementos automáticamente!

¡Es responsabilidad del programador usar índices válidos!

No se pueden copiar arrays directamente (~~array1 = array2~~)

Hay que copiarlos elemento a elemento



# Arrays y bucles for

## Procesamiento de arrays

- ✓ Recorridos
  - ✓ Búsquedas
  - ✓ Ordenación
- etcétera...

## Recorrido de arrays con bucles for

Arrays: tamaño fijo → Bucles de recorrido fijo (for)

```
tVentas ventas;  
double media, total = 0;  
...  
for (int i = 0; i < Dias; i++) {  
    total = total + ventas[i];  
}  
media = total / Dias;
```

```
const int Dias = 7;  
typedef double tVentas[Dias];
```

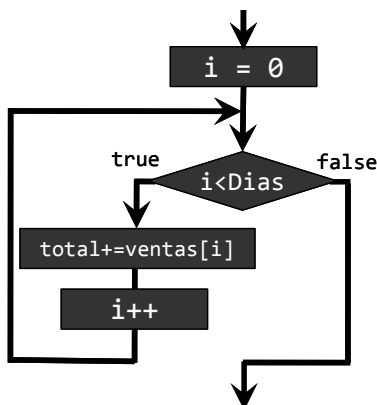
Luis Hernández Yáñez/Pablo Moreno Ger



# Arrays y bucles for

12.40	10.96	8.43	11.65	13.70	13.41	14.07
0	1	2	3	4	5	6

```
tVentas ventas;  
double media, total = 0;  
...  
for (int i = 0; i < Dias; i++) {  
    total = total + ventas[i];  
}
```



	Memoria
Dias	7
→ ventas[0]	12.40
→ ventas[1]	10.96
→ ventas[2]	8.43
→ ventas[3]	11.65
→ ventas[4]	13.70
→ ventas[5]	13.41
→ ventas[6]	14.07
media	?
total	84.62
i	7

Luis Hernández Yáñez/Pablo Moreno Ger



## Más sobre arrays



## Inicialización de arrays

---

Podemos inicializar los elementos de los arrays en la declaración  
*Asignamos* una serie de valores al array:

```
const int DIM = 10;  
typedef int tTabla[DIM];  
tTabla i = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Se asignan los valores por su orden:

<code>i[0]</code>	<code>i[1]</code>	<code>i[2]</code>	<code>i[3]</code>	<code>i[4]</code>	<code>...</code>	<code>i[9]</code>
↑	↑	↑	↑	↑		↑
1º	2º	3º	4º	5º	...	10º

Si hay menos valores que elementos, los restantes se ponen a 0

```
tTabla i = { 0 }; // Pone todos los elementos a 0
```



# Enumerados como índices

```
const int Colores = 3,  
typedef enum { rojo, verde, azul } tRGB;  
typedef int tColor[Colores];  
tColor color;  
...  
cout << "Cantidad de rojo (0-255): ";  
cin >> color(rojo);  
cout << "Cantidad de verde (0-255): ";  
cin >> color[verde];  
cout << "Cantidad de azul (0-255): ";  
cin >> color[azul];
```

Recuerda que internamente se asignan enteros a partir de 0  
a los distintos símbolos del enumerado  
rojo  $\equiv$  0 verde  $\equiv$  1 azul  $\equiv$  2



# Paso de arrays a subprogramas

*Simulación de paso de parámetro por referencia*

Sin poner & en la declaración del parámetro

Los subprogramas reciben la dirección en memoria del array

```
const int Max = 10;  
typedef int tTabla[Max];  
void inicializa(tTabla tabla); // Sin poner &
```

Las modificaciones del array quedan reflejadas en el argumento

```
inicializa(array);
```

Si `inicializa()` modifica algún elemento de `tabla`,  
automáticamente queda modificado ese elemento de array

*¡Son el mismo array!*



# Paso de arrays a subprogramas

```
const int Dim = 10;
typedef int tTabla[Dim];
void inicializa(tTabla tabla); // no se usa &

void inicializa(tTabla tabla) {
    for (int i = 0; i < Dim; i++)
        tabla[i] = i;
}
int main() {
    tTabla array;
    inicializa(array); // array queda modificado
    for (int i = 0; i < Dim; i++)
        cout << array[i] << " ";
    ...
}
```

0 1 2 3 4 5 6 7 8 9



# Paso de arrays a subprogramas

*¿Cómo evitar que se modifique el array?*

Usando el modificador `const` en la declaración del parámetro:

```
const tTabla tabla    Un array de constantes
```

```
void muestra(const tTabla tabla);
```

El argumento se tratará como un array de constantes

Si en el subprograma hay alguna instrucción que intente modificar un elemento del array: error de compilación

```
void muestra(const tTabla tabla) {
    for (int i = 0; i < Dim; i++) {
        cout << tabla[i] << " ";
        // OK. Se accede, pero no se modifica
    }
}
```





## Implementación de listas



## Implementación de listas con arrays

---

### *Listas con un número fijo de elementos*

Array con el  $n^{\circ}$  de elementos como dimensión

```
const int NUM = 100;  
typedef double tLista[NUM]; // Exactamente 100 double  
tLista lista;
```

Recorrido de la lista:

```
for (int i = 0; i < NUM; i++) {  
    ...
```

Búsqueda en la lista:

```
while ((i < NUM) && !encontrado) {  
    ...
```



# Implementación de listas con arrays

---

## *Listas con un número variable de elementos*

Array con un máximo de elementos + Contador de elementos

```
const int MAX = 100;
typedef double tLista[MAX]; // Hasta 100 elementos
tLista lista;
int contador = 0; // Se incrementa al insertar

Recorrido de la lista:
for (int i = 0; i < contador; i++) {
    ...
}

Búsqueda en la lista:
while ((i < contador) && !encontrado) {
    ...
}
```

¿Array y contador por separado? → Estructuras



# Fundamentos de la programación

---

## Cadenas de caracteres



# Cadenas de caracteres

## Arrays de caracteres

Cadenas: secuencias de caracteres de longitud variable

"Hola" "Adiós" "Supercalifragilístico" "1234 56 7"

Variables de cadena: contienen secuencias de caracteres

Se guardan en arrays de caracteres: tamaño máximo (dimensión)

No todas las posiciones del array son relevantes:

- ✓ Longitud de la cadena: número de caracteres, desde el primero, que realmente constituyen la cadena:

H	o	l	a																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	

Longitud actual: 4



# Cadenas de caracteres

## Longitud de la cadena

A	d	i	ó	s																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	

Longitud: 5

S	u	p	e	r	c	a	l	i	f	r	a	g	i	l	í	s	t	i	c	o		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	

Longitud: 21

Necesidad de saber dónde terminan los caracteres relevantes:

- ✓ Mantener la longitud de la cadena como dato asociado
- ✓ Colocar un carácter de terminación al final (*centinela*)

A	d	i	ó	s	\0																	
0	1	2	3	4	5	6	7	8	9	10												



# Cadenas de caracteres

---

## *Cadenas de caracteres en C++*

Dos alternativas para el manejo de cadenas:

- ✓ Cadenas al estilo de C (*terminadas en nulo*)
- ✓ Tipo `string`

Cadenas al estilo de C

*Anexo del tema*

- ✓ Arrays de tipo `char` con una longitud máxima
- ✓ Un último carácter especial al final: `'\0'`

Tipo `string`

- ✓ Cadenas más sofisticadas
- ✓ Sin longitud máxima (gestión automática de la memoria)
- ✓ Multitud de funciones de utilidad (biblioteca `string`)



# Fundamentos de la programación

---

## Cadenas de caracteres de tipo `string`



# Cadenas de caracteres de tipo string

## El tipo string

- ✓ El tipo asume la responsabilidad de la gestión de memoria
- ✓ Define operadores sobrecargados (+ para concatenar)
- ✓ Cadenas más eficientes y seguras de usar

## Biblioteca string

Requiere establecer el espacio de nombres a std

- ✓ Se pueden inicializar en la declaración
- ✓ Se pueden copiar con el operador de asignación
- ✓ Se pueden concatenar con el operador +
- ✓ Multitud de funciones de utilidad



# Cadenas de tipo string

string.cpp

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

int main() {
    string cad1("Hola"); // inicialización
    string cad2 = "amigo"; // inicialización
    string cad3;
    cad3 = cad1; // copia
    cout << "cad3 = " << cad3 << endl;
    cad3 = cad1 + " "; // concatenación
    cad3 += cad2; // concatenación
    cout << "cad3 = " << cad3 << endl;
    cad1.swap(cad2); // intercambio
    cout << "cad1 = " << cad1 << endl;
    cout << "cad2 = " << cad2 << endl;

    return 0;
}
```

```
Simbolo del sistema
D:\FP\Tema5>string
cad3 = Hola
cad3 = Hola amigo
cad1 = amigo
cad2 = Hola
D:\FP\Tema5>
```



# Cadenas de tipo string

---

Longitud de la cadena:

`cadena.length()` o `cadena.size()`

Se pueden comparar con los operadores relacionales:

`if (cad1 <= cad2) { ...`

Acceso a los caracteres de una cadena:

✓ Como array de caracteres: `cadena[i]`

Sin control de acceso a posiciones inexistentes del array

Sólo debe usarse si se está seguro de que el índice es válido

✓ Función `at(índice)`: `cadena.at(i)`

Error de ejecución si se accede a una posición inexistente



# E/S con cadenas de tipo string

---

✓ Se muestran en la pantalla con `cout <<`

✓ Lectura con `cin >>`: termina con espacio en blanco (inc. Intro)

El espacio en blanco queda pendiente

✓ Descartar el resto de los caracteres del búfer:

`cin.sync();`

✓ Lectura incluyendo espacios en blanco:

`getline(cin, cadena)`

Guarda en la *cadena* los caracteres leídos hasta el fin de línea

✓ Lectura de archivos de texto:

Igual que de consola; `sync()` no tiene efecto

`archivo >> cadena`      `getline(archivo, cadena)`



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

int main() {
    string nombre, apellidos;
    cout << "Introduzca un nombre: ";
    cin >> nombre;
    cout << "Introduzca los apellidos: ";
    cin.sync();
    getline(cin, apellidos);
    cout << "Nombre completo: " << nombre << " "
        << apellidos << endl;

    return 0;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger



## Operaciones con cadenas de tipo string

- ✓ *cadena.substr(posición, Longitud)*  
Subcadena de *longitud* caracteres desde *posición*  
string cad = "abcdefg";  
cout << cad.substr(2, 3); // Muestra cde
- ✓ *cadena.find(subcadena)*  
Posición de la primera ocurrencia de *subcadena* en *cadena*  
string cad = "Olala";  
cout << cad.find("la"); // Muestra 1  
(Recuerda que los arrays de caracteres comienzan con el índice 0)
- ✓ *cadena.rfind(subcadena)*  
Posición de la última ocurrencia de *subcadena* en *cadena*  
string cad = "Olala";  
cout << cad.rfind("la"); // Muestra 3

Luis Hernández Yáñez/Pablo Moreno Ger



# Operaciones con cadenas de tipo string

---

✓ `cadena.erase(ini, num)`

Elimina *num* caracteres a partir de la posición *ini*

```
string cad = "abcdefgh";  
cad.erase(3, 4); // cad ahora contiene "abch"
```

✓ `cadena.insert(ini, cadena2)`

Inserta *cadena2* a partir de la posición *ini*

```
string cad = "abcdefgh";  
cad.insert(3, "123"); // cad ahora contiene "abc123defgh"
```

<http://www.cplusplus.com/reference/string/string/>



# Fundamentos de la programación

---

## Estructuras





# Estructuras

---

## *Colecciones heterogéneas (tuplas, registros)*

Elementos de (posiblemente) distintos tipos: *campos*

Campos identificados por su nombre

*Información relacionada que se puede manejar como una unidad*

Acceso a cada elemento por su nombre de campo (operador .)



## Tipos de estructuras

---

```
typedef struct {  
    ... // declaraciones de campos (como variables)  
} tTipo; // nombre de tipo - ¡al final!
```

```
typedef struct {  
    string nombre;  
    string apellidos;  
    int edad;  
    string nif;  
} tPersona;
```

Campos:

Tipos estándar o previamente declarado



# Variables de estructuras

```
tPersona persona;
```

Las variables de tipo tPersona contienen cuatro datos (campos):

```
nombre    apellidos    edad    nif
```

Acceso a los campos con el operador punto (.):

```
persona.nombre // una cadena (string)
```

```
persona.apellidos // una cadena (string)
```

```
persona.edad // un entero (int)
```

```
persona.nif // una cadena (string)
```

Podemos copiar dos estructuras directamente:

```
tPersona persona1, persona2;
```

```
...
```

```
persona2 = persona1;
```

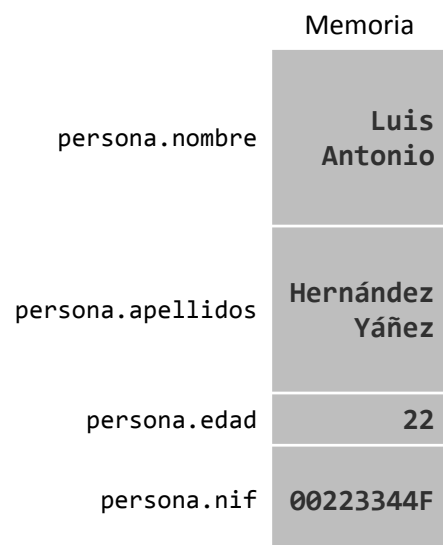
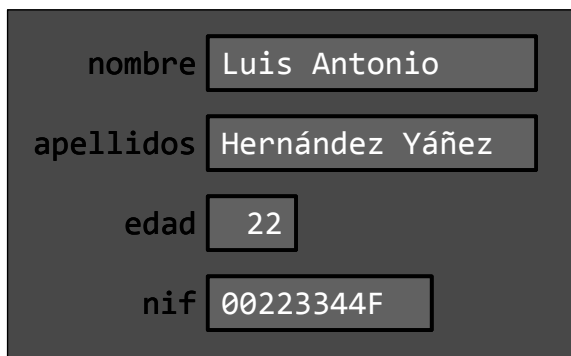
Se copian todos los campos a la vez



# Agrupación de datos heterogéneos

```
typedef struct {  
    string nombre;  
    string apellidos;  
    int edad;  
    string nif;  
} tPersona;  
tPersona persona;
```

persona



# Elementos sin orden establecido

```
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
tPersona persona;
```

Los campos no siguen ningún orden establecido  
Acceso directo por nombre de campo (operador .)  
Con cada campo se puede hacer lo que permita su tipo

Las estructuras se pasan por valor (sin &) o por referencia (con &) a los subprogramas



# Estructuras dentro de estructuras

```
typedef struct {
    string dni;
    char letra;
} tNif;

typedef struct {
    ...
    tNif nif;
} tPersona;
```

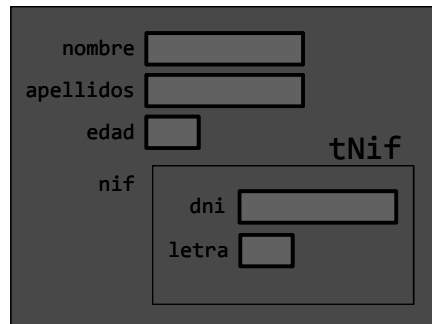
tPersona persona;

Acceso al NIF completo:  
persona.nif // Otra estructura

Acceso a la letra del NIF:  
persona.nif.letra

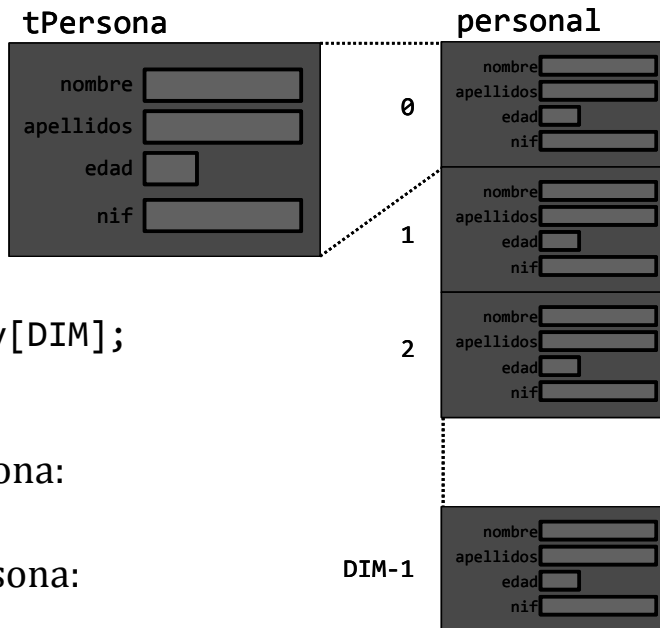
Acceso al DNI:  
persona.nif.dni

tPersona



# Arrays de estructuras

```
const int DIM = 100;  
typedef struct {  
    string nombre;  
    string apellidos;  
    int edad;  
    string nif;  
} tPersona;  
typedef tPersona tArray[DIM];  
tArray personal;
```



Nombre de la tercera persona:

`personal[2].nombre`

Edad de la duodécima persona:

`personal[11].edad`

NIF de la primera persona:

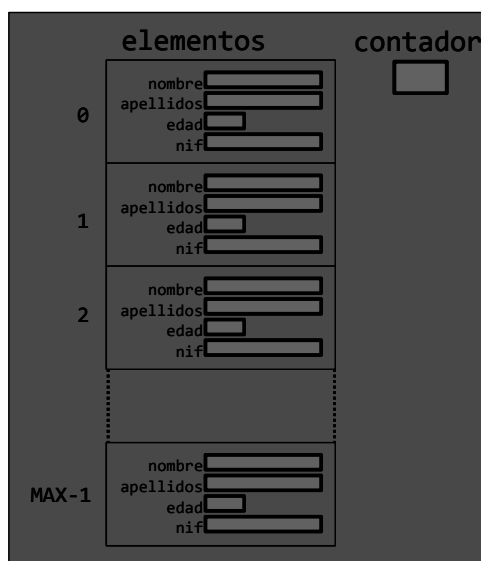
`personal[0].nif`

Luis Hernández Yáñez/Pablo Moreno Ger



# Arrays dentro de estructuras

```
const int MAX = 100; lista  
typedef struct {  
    string nombre;  
    string apellidos;  
    int edad;  
    string nif;  
} tPersona;  
typedef tPersona tArray[MAX];  
typedef struct {  
    tArray elementos;  
    int contador;  
} tLista;  
tLista lista;
```



Nombre de la tercera persona: `lista.elementos[2].nombre`

Edad de la duodécima persona: `lista.elementos[11].edad`

NIF de la primera persona: `lista.elementos[0].nif`

Luis Hernández Yáñez/Pablo Moreno Ger



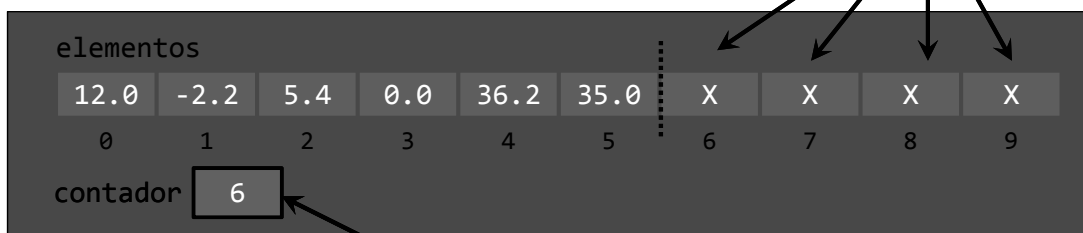
## Listas de longitud variable



## Listas de longitud variable

Estructura que agrupe el array y el contador:

```
const int MAX = 10;  
typedef double tArray[MAX];  
typedef struct {  
    tArray elementos;  
    int contador;  
} tLista;
```



Nº de elementos (y primer índice sin elemento)

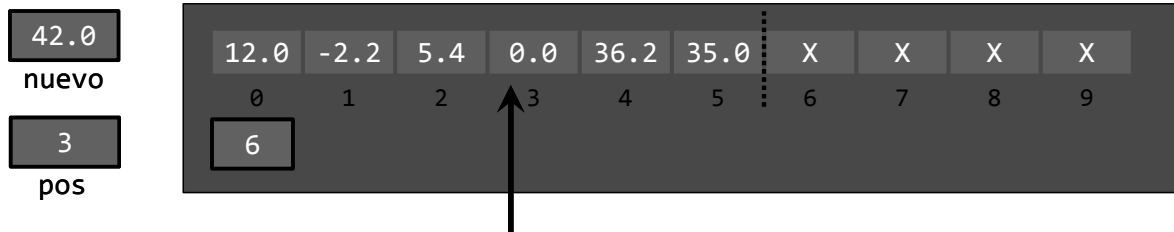
Operaciones principales: inserción y eliminación de elementos



# Inserción de elementos

## *Insertar un nuevo elemento en una posición*

Posiciones válidas: 0 a contador



Hay que asegurarse de que haya sitio (contador < máximo)

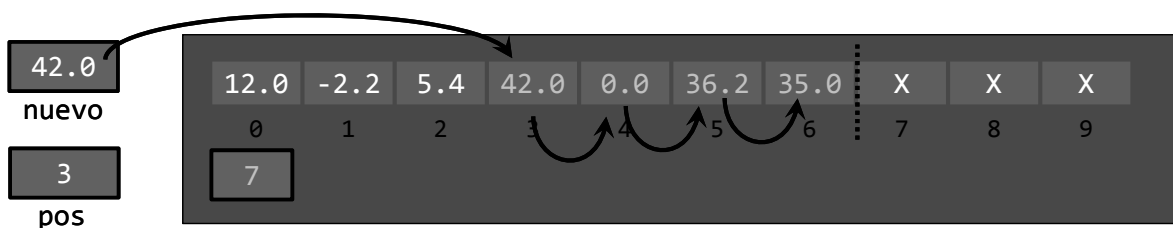
Operación en 3 pasos:

- 1.- Abrir hueco para el nuevo elemento (desde la posición)
- 2.- Colocar el elemento nuevo en la posición
- 3.- Incrementar el contador



# Inserción de elementos

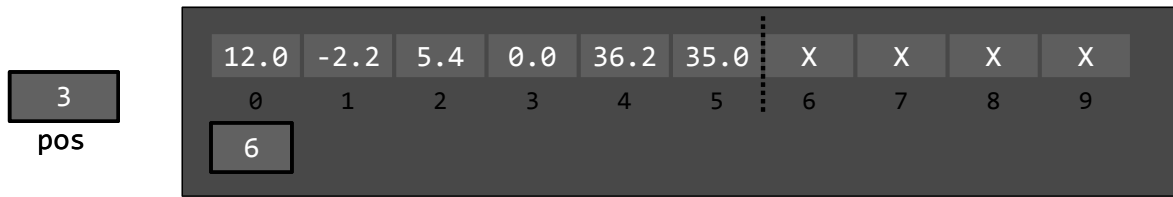
```
if (lista.contador < N) {  
    // Abrir hueco  
    for (int i = lista.contador; i > pos; i--) {  
        lista.elementos[i] = lista.elementos[i - 1];  
    }  
    // Insertar e incrementar contador  
    lista.elementos[pos] = nuevoElemento;  
    lista.contador++;  
}
```



# Eliminación de elementos

## *Eliminar el elemento en una posición*

Posiciones válidas: 0 a contador-1



Desplazar a la izquierda desde el siguiente y decrementar el contador:

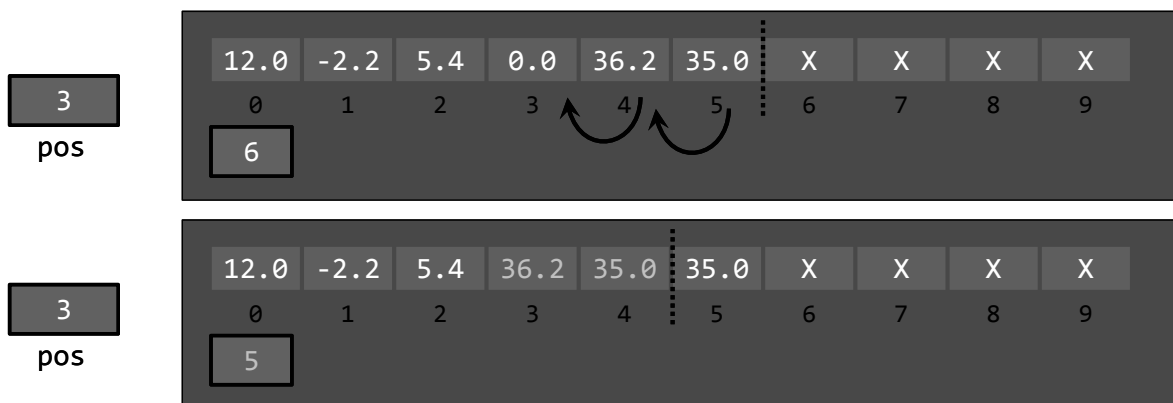
```
for (int i = pos; i < lista.contador - 1 ; i++) {  
    lista.elementos[i] = lista.elementos[i + 1];  
}  
lista.contador--;
```

Luis Hernández Yáñez/Pablo Moreno Ger



# Eliminación de elementos

```
for (int i = pos; i < lista.contador - 1 ; i++) {  
    lista.elementos[i] = lista.elementos[i + 1];  
}  
lista.contador--;
```



Luis Hernández Yáñez/Pablo Moreno Ger



## Un ejemplo completo



## Ejemplo de lista de longitud variable

---

### *Descripción*

Programa que mantenga una lista de los estudiantes de una clase

De cada estudiante: nombre, apellidos, edad, NIF y nota

- ✓ Se desconoce el número total de estudiantes (máximo 100)
- ✓ La información de la lista se mantiene en un archivo `clase.txt`  
Se carga al empezar y se guarda al finalizar
- ✓ El programa debe ofrecer estas opciones:
  - Añadir un nuevo alumno
  - Eliminar un alumno existente
  - Calificar a los estudiantes
  - Listado de notas, identificando la mayor y la media





# Ejemplo de lista de longitud variable

bd.cpp

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

const int MAX = 100;
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
    double nota;
} tEstudiante;
typedef tEstudiante tArray[MAX];
typedef struct {
    tArray elementos;
    int contador;
} tLista;
```

Declaraciones de constantes  
y tipos globales  
Tras las bibliotecas

Luis Hernández Yáñez/Pablo Moreno Ger



Fundamentos de la programación: Tipos de datos estructurados

Página 560



# Ejemplo de lista de longitud variable

```
// Prototipos
int menu(); // Menú del programa - devuelve la opción elegida
void cargar(tLista &lista, bool &ok); // Carga del archivo
void guardar(const tLista &lista); // La guarda en el archivo
void leerEstudiante(tEstudiante &estudiante); // Lee los datos
void insertarEstudiante(tLista &lista, tEstudiante estudiante,
    bool &ok); // Inserta un nuevo estudiante en la lista
void eliminarEstudiante(tLista &lista, int pos, bool &ok);
// Elimina el estudiante en esa posición
string nombreCompleto(tEstudiante estudiante);
void calificar(tLista &lista); // Notas de los estudiantes
double mediaClase(const tLista &lista); // Nota media
int mayorNota(const tLista &lista);
// Índice del estudiante con mayor nota
void mostrarEstudiante(tEstudiante estudiante);
void listado(const tLista &lista, double media, int mayor);
// Listado de la clase
```

Los prototipos, después de los tipos globales

Luis Hernández Yáñez/Pablo Moreno Ger



Fundamentos de la programación: Tipos de datos estructurados

Página 561



## El bucle do-while

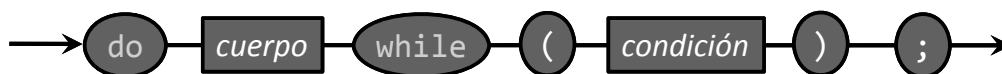


## Otro bucle no determinado de C++

---

### *El bucle do..while*

do  *cuerpo*  while (*condición*);      Condición al final del bucle



```
int i = 1;
do {
    cout << i << endl;
    i++;
} while (i <= 100);
```

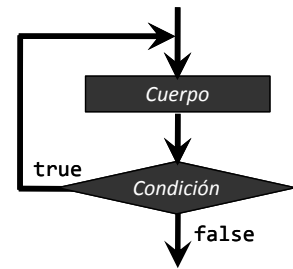
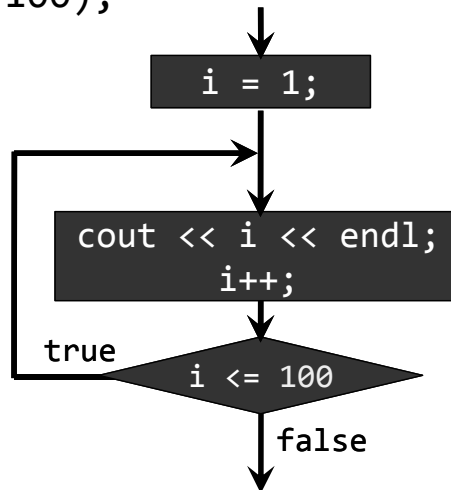
El *cuerpo* siempre se ejecuta al menos una vez

El *cuerpo* es un bloque de código



# Ejecución del bucle do-while

```
int i = 1;
do {
    cout << i << endl;
    i++;
} while (i <= 100);
```



El cuerpo se ejecuta al menos una vez

Luis Hernández Yáñez/Pablo Moreno Ger



# while versus do-while

*¿Ha de ejecutarse al menos una vez el cuerpo del bucle?*

```
cin >> d; // Lectura del 1º
while (d != 0) {
    suma = suma + d;
    cont++;
    cin >> d;
}
```

```
do {
    cin >> d;
    if (d != 0) { // ¿Final?
        suma = suma + d;
        cont++;
    }
} while (d != 0);
```

```
cout << "Opción: ";
cin >> op; // Lectura del 1º
while ((op < 0) || (op > 4)) {
    cout << "Opción: ";
    cin >> op;
}
```

```
do { // Más simple
    cout << "Opción: ";
    cin >> op;
} while ((op < 0) || (op > 4));
```

Luis Hernández Yáñez/Pablo Moreno Ger



# El menú de la aplicación con do-while

```
int menu() {
    int op;

    do {
        cout << "1 - Añadir un nuevo estudiante" << endl;
        cout << "2 - Eliminar un estudiante" << endl;
        cout << "3 - Calificar a los estudiantes" << endl;
        cout << "4 - Listado de estudiantes" << endl;
        cout << "0 - Salir" << endl;
        cout << "Opción: ";
        cin >> op;
    } while ((op < 0) || (op > 4));

    return op;
}
```



# Ejemplo de lista de longitud variable

## El archivo clase.txt

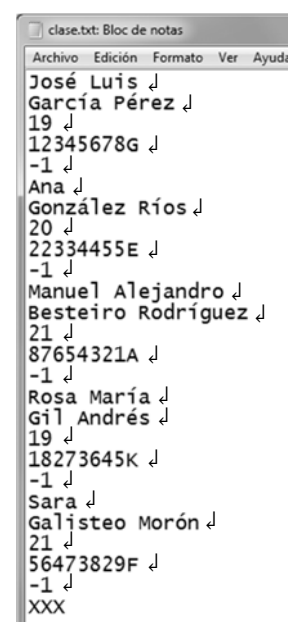
Un dato en cada línea

Por cada estudiante:

- ✓ Nombre (cadena)
- ✓ Apellidos (cadena)
- ✓ Edad (entero)
- ✓ NIF (cadena)
- ✓ Nota (real; -1 si no calificado)

Termina con XXX como nombre

El archivo se supone correcto



# Ejemplo de lista de longitud variable

## *Lectura de la información de un estudiante*

Nombre y apellidos:

Puede haber varias palabras → `getline()`

Edad → extractor (`>>`)

NIF: Una sola palabra → extractor (`>>`)

Nota → extractor (`>>`)

Queda pendiente de leer el Intro

Hay que saltar (leer) ese carácter con `get()`

Si no, en el siguiente nombre se leería una cadena vacía (Intro)



No leas directamente en la lista:

```
getline(archivo, lista.elementos[lista.contador].nombre);
```

Lee en una variable auxiliar de tipo `tEstudiante`



## Carga del archivo `clase.txt`

```
void cargar(tLista &lista, bool &ok) {
    tEstudiante estudiante; // Variable auxiliar para leer
    ifstream archivo;
    char aux;
    lista.contador = 0; // Inicializamos la lista
    archivo.open("clase.txt");
    if (!archivo.is_open()) {
        ok = false;
    }
    else {
        ok = true;
        getline(archivo, estudiante.nombre); // Leemos el primer nombre
        while ((estudiante.nombre != "XXX") && (lista.contador < MAX)) {
            getline(archivo, estudiante.apellidos);
            archivo >> estudiante.edad;
            archivo >> estudiante.nif;
            archivo >> estudiante.nota;
            archivo.get(aux); // Saltamos el Intro
            → lista.elementos[lista.contador] = estudiante; // Al final
              lista.contador++;
            getline(archivo, estudiante.nombre); // Siguiete nombre
        } // Si hay más de MAX estudiantes, ignoramos el resto
        archivo.close();
    }
}
```



# Volcado en el archivo clase.txt

Simplemente, un dato en cada línea y en orden:

```
void guardar(const tLista &lista) {
    ofstream archivo;
    archivo.open("clase.txt");
    for (int i = 0; i < lista.contador; i++) {
        archivo << lista.elementos[i].nombre << endl;
        archivo << lista.elementos[i].apellidos << endl;
        archivo << lista.elementos[i].edad << endl;
        archivo << lista.elementos[i].nif << endl;
        archivo << lista.elementos[i].nota << endl;
    }
    archivo << "XXX" << endl; // Centinela final
    archivo.close();
}
```

`const tLista &lista` → Referencia constante  
Paso por referencia pero como constante ≡ Paso por valor  
Evita la copia del argumento en el parámetro (estructuras grandes)



# Lectura de los datos de un estudiante

```
void leerEstudiante(tEstudiante &estudiante) {
    cin.sync(); // Descartamos cualquier entrada pendiente
    cout << "Nombre: ";
    getline(cin, estudiante.nombre);
    cout << "Apellidos: ";
    getline(cin, estudiante.apellidos);
    cout << "Edad: ";
    cin >> estudiante.edad;
    cout << "NIF: ";
    cin >> estudiante.nif;
    estudiante.nota = -1; // Sin calificar de momento
    cin.sync(); // Descartamos cualquier entrada pendiente
}
```



# Inserción de un nuevo estudiante

---

```
void insertarEstudiante(tLista &lista, tEstudiante estudiante,
    bool &ok) {

    ok = true;
    if (lista.contador == MAX) {
        ok = false;
    }
    else {
        lista.elementos[lista.contador] = estudiante;
        // Insertamos al final
        lista.contador++;
    }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger



# Eliminación de un estudiante

---

```
void eliminarEstudiante(tLista &lista, int pos, bool &ok) {
    // Espera el índice del elemento en pos

    if ((pos < 0) || (pos > lista.contador - 1)) {
        ok = false; // Elemento inexistente
    }
    else {
        ok = true;
        for (int i = pos; i < lista.contador - 1; i++) {
            lista.elementos[i] = lista.elementos[i + 1];
        }
        lista.contador--;
    }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger



# Calificación de los estudiantes

---

```
string nombreCompleto(tEstudiante estudiante) {
    return estudiante.nombre + " " + estudiante.apellidos;
}

void calificar(tLista &lista) {

    for (int i = 0; i < lista.contador; i++) {
        cout << "Nota del estudiante "
            << nombreCompleto(lista.elementos[i]) << ": ";
        cin >> lista.elementos[i].nota;
    }
}
```



## Más subprogramas

---

```
double mediaClase(const tLista &lista) {
    double total = 0.0;
    for (int i = 0; i < lista.contador; i++) {
        total = total + lista.elementos[i].nota;
    }
    return total / lista.contador;
}

int mayorNota(const tLista &lista) {
    double max = 0;
    int pos = 0;
    for (int i = 0; i < lista.contador; i++) {
        if (lista.elementos[i].nota > max) {
            max = lista.elementos[i].nota;
            pos = i;
        }
    }
    return pos;
}
```





# El listado

---

```
void mostrarEstudiante(tEstudiante estudiante) {
    cout << setw(35) << left << nombreCompleto(estudiante);
    cout << estudiante.nif << " ";
    cout << setw(2) << estudiante.edad << " años ";
    cout << fixed << setprecision(1) << estudiante.nota;
}

void listado(const tLista &lista, double media, int mayor) {
    for (int i = 0; i < lista.contador; i++) {
        cout << setw(3) << i << ": ";
        mostrarEstudiante(lista.elementos[i]);
        if (i == mayor) {
            cout << " <<< Mayor nota!";
        }
        cout << endl;
    }
    cout << "Media de la clase: " << fixed << setprecision(1)
        << media << endl << endl;
}
```



# El programa principal

---

```
int main() {
    tLista lista;
    tEstudiante estudiante;
    bool exito;
    int op, pos;

    cargar(lista, exito);
    if (!exito) {
        cout << "No se ha podido cargar la lista!" << endl;
    }
    else {
        do { // El bucle do evita tener que leer antes la primera opción
            op = menu();
            switch (op) {
                case 1:
                    {
                        leerEstudiante(estudiante);
                        insertarEstudiante(lista, estudiante, exito);
                        if (!exito) {
                            cout << "Lista llena: imposible insertar" << endl;
                        }
                    }
                }
            }
            break;
        }
    }
}
```



# El programa principal

```
case 2:
{
    cout << "Posición: ";
    cin >> pos;
    eliminarEstudiante(lista, pos - 1, exito);
    if (!exito) {
        cout << "Elemento inexistente!" << endl;
    }
}
break;
case 3:
{
    calificar(lista);
}
break;
case 4:
{
    listado(lista, mediaClase(lista), mayorNota(lista));
}
} while (op != 0);
guardar(lista);
}
return 0;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger






## Acerca de *Creative Commons*



### *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

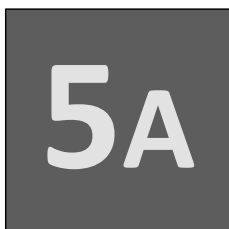
Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.

Luis Hernández Yáñez/Pablo Moreno Ger





ANEXO

## Cadenas de caracteres al estilo de C

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez/Pablo Moreno Ger  
Facultad de Informática  
Universidad Complutense



## Índice

---

Cadenas al estilo de C	582
E/S con cadenas al estilo de C	583
La biblioteca <code>cstring</code>	584
Ejemplo	585



# Cadenas de caracteres al estilo de C

## Arrays de caracteres terminados en nulo

```
const Max = 15;
typedef char tCadena[Max];
tCadena cadena = "Adiós"; // Inicialización al declarar
Siempre hay al final un carácter nulo (código ASCII 0 - '\0')
Indica que en esa posición termina la cadena (exclusive)
```

cadena	A	d	i	ó	s	\0											
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14		

En el array caben MAX-1 caracteres significativos  
Longitud máxima de la variable cadena: 14

No se pueden asignar cadenas literales: ~~cadena = "Hola";~~  
Ni copiar cadenas directamente: ~~cad2 = cad1;~~  
Ni comparar con op. relacionales: ~~if (cad1 < cad2) ...~~

Luis Hernández Yáñez/Pablo Moreno Ger



# Entrada/salida con cadenas al estilo de C

```
tCadena cadena;
cin >> cadena; // Se añade un nulo al final
Extractor: la lectura termina en el primer espacio en blanco
¡No se comprueba si se leen más caracteres de los que caben!
setw(): máximo de caracteres a colocar (incluyendo el nulo)
cin >> setw(15) >> cadena;
cin.getline(cadena_estilo_C, máx):
Para leer también los espacios en blanco y no más de máx-1
cin.getline(cadena, 15); // Hasta 14 caracteres
cout << cadena << endl; // El nulo no se muestra
```

	cin.getline(cad, máx)	Cadenas al estilo de C
	getline(cin, cad)	Cadenas de tipo string

Luis Hernández Yáñez/Pablo Moreno Ger



# La biblioteca cstring

- ✓ `strlen(cadena)`: longitud actual de la *cadena*  
`cout << "Longitud: " << strlen(cadena);`
- ✓ `strcpy(destino, origen)`: copia *origen* en *destino*  
`strcpy(cad2, cad1); strcpy(cad, "Me gusta C++");`
- ✓ `strcat(destino, origen)`: añade *origen* al final de *destino*  
`tCadena cad1 = "Hola", cad2 = "Adiós";`  
`strcat(cad1, cad2); // cad1 contiene "HolaAdiós"`
- ✓ `strcmp(cad1, cad2)`: compara lexicográficamente las cadenas  
0 si son iguales, 1 si *cad1* > *cad2* ó -1 si *cad1* < *cad2*  
`tCadena cad1 = "Hola", cad2 = "Adiós";`  
`strcmp(cad1, cad2) // Devuelve 1 ("Hola" > "Adiós")`

...

<http://www.cplusplus.com/reference/cstring/>



## Ejemplo de cadenas al estilo de C

`cadenas.cpp`

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

int main() {
    const int MAX = 20;
    typedef char tCad[MAX];
    tCad cadena = "Me gusta C++";
    cout << cadena << endl;
    cout << "Cadena: ";
    cin >> cadena; // Lee hasta el primer espacio en blanco
    cout << cadena << endl;
    cin.sync(); // Sincronizar la entrada
    cout << "Cadena: ";
    cin.getline(cadena, MAX);
    cout << cadena << endl;
    cout << "Longitud: " << strlen(cadena) << endl;
    strcpy(cadena, "Hola");
    ...
}
```



# Ejemplo de cadenas al estilo de C

```
tCad cadena2 = " amigo";
strcat(cadena, cadena2);
cout << cadena << endl;
if (strcmp(cadena, cadena2) == 0) {
    cout << "Iguales";
}
else if (strcmp(cadena, cadena2) > 0) {
    cout << cadena << " es mayor que " << cadena2;
}
else {
    cout << cadena << " es menor que " << cadena2;
}
cout << endl;

return 0;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger






## Acerca de *Creative Commons*



### *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.

Luis Hernández Yáñez/Pablo Moreno Ger



## 6

# Recorrido y búsqueda en arrays

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez / Pablo Moreno Ger  
Facultad de Informática  
Universidad Complutense



## Índice

---

Recorrido de arrays	590
Arrays completos	593
Arrays no completos con centinela	594
Arrays no completos con contador	595
Ejemplos	597
Generación de números aleatorios	601
Búsquedas en arrays	604
Arrays completos	606
Arrays no completos con centinela	607
Arrays no completos con contador	608
Ejemplo	610
Recorridos y búsquedas en cadenas	614
Más ejemplos de manejo de arrays	617
Arrays multidimensionales	630
Inicialización de arrays multidimensionales	638
Recorrido de un array bidimensional	641
Recorrido de un array N-dimensional	644
Búsqueda en un array multidimensional	647



## Recorrido de arrays

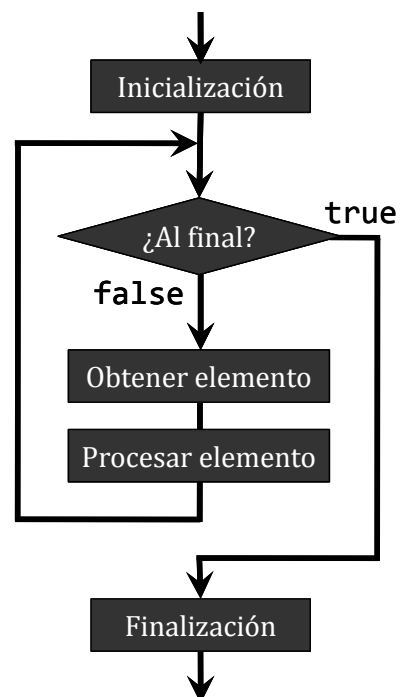
Luis Hernández Yáñez



## Recorrido de arrays

### *Esquema de recorrido*

- Inicialización
- Mientras no al final de la secuencia:
  - Obtener el siguiente elemento
  - Procesar el elemento
- Finalización



Luis Hernández Yáñez





# Recorrido de arrays

---

## *Recorrido de secuencias en arrays*

- ✓ Todas las posiciones ocupadas:
  - Tamaño del array = longitud de la secuencia
  - N elementos en un array de N posiciones:
  - Recorrer el array desde la primera posición hasta la última
- ✓ Posiciones libres al final del array:
  - Tamaño del array > longitud de la secuencia
  - Con centinela:
    - Recorrer el array hasta encontrar el valor centinela
  - Con *contador* de elementos:
    - Recorrer el array hasta el índice *contador* - 1



# Recorrido de arrays

---

## *Recorrido de arrays completos*

Todas las posiciones del array ocupadas

```
const int N = 10;
typedef double tVentas[N];
tVentas ventas;
...
ventas
  125.40  76.95  328.80  254.62  435.00  164.29  316.05  219.99  93.45  756.62
    0      1      2      3      4      5      6      7      8      9

double elemento;
for (int i = 0; i < N; i++) {
    elemento = ventas[i];
    // Procesar el elemento ...
}
```



# Recorrido de arrays

## Recorrido de arrays no completos – con centinela

No todas las posiciones del array están ocupadas

```
const int N = 10;
typedef double tArray[N];
tArray datos; // Datos positivos: centinela = -1
...
```

datos	125.40	76.95	328.80	254.62	435.00	164.29	316.05	-1.0		
	0	1	2	3	4	5	6	7	8	9

```
int i = 0;
double elemento = datos[i];
while (elemento != -1) {
    // Procesar el elemento ...
    i++;
    elemento = datos[i];
}
```

```
int i = 0;
double elemento;
do {
    elemento = datos[i];
    if (elemento != -1) {
        // Procesar el elemento...
        i++;
    }
} while (elemento != -1);
```

Luis Hernández Yáñez



# Recorrido de arrays

## Recorrido de arrays no completos – con contador

Array y contador íntimamente relacionados: estructura

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elementos;
    int contador;
} tLista;
```

Listas de elementos de longitud variable

elementos	125.40	76.95	328.80	254.62	435.00	164.29	316.05			
	0	1	2	3	4	5	6	7	8	9
contador	7									

Nº de elementos (primer índice sin elemento)

Luis Hernández Yáñez



# Recorrido de arrays

---

## *Recorrido de arrays no completos – con contador*

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elementos;
    int contador;
} tLista;
tLista lista;

...
double elemento;
for (int i = 0; i < lista.contador; i++) {
    elemento = lista.elementos[i];
    // Procesar el elemento...
}
```



# Fundamentos de la programación

---

## Ejemplos



## Array con los N primeros números de Fibonacci

```
const int N = 50;
typedef long long int tFibonacci[N]; // 50 números
tFibonacci fib;
fib[0] = 1;
fib[1] = 1;
for (int i = 2; i < N; i++) {
    fib[i] = fib[i - 1] + fib[i - 2];
}
for (int i = 0; i < N; i++) {
    cout << fib[i] << " ";
}
```

Luis Hernández Yáñez



# Ejemplos

## Cuenta de valores con k dígitos

Recorrer una lista de N enteros contabilizando cuántos son de 1 dígito, cuántos de 2 dígitos, etcétera (hasta 5 dígitos)

2 arrays: array con los números y array de contadores

```
const int NUM = 100;
typedef int tNum[NUM]; // Exactamente 100 números
tNum numeros;
const int DIG = 5;
typedef int tDig[DIG]; // i --> números de i+1 dígitos
tDig numDig = { 0 };
```

numeros	123	2	46237	2345	236	11234	33	999	...	61
	0	1	2	3	4	5	6	7		99
numDig	0	0	0	0	0	0				
	0	1	2	3	4	5				

Luis Hernández Yáñez



# Ejemplos

---

## *Cuenta de valores con k dígitos*

Función que devuelve el número de dígitos de un entero:

```
int digitos(int dato) {
    int n_digitos = 1; // Al menos tiene un dígito
    // Recorremos la secuencia de dígitos...
    while (dato >= 10) {
        dato = dato / 10;
        n_digitos++;
    }
    return n_digitos;
}
```

Luis Hernández Yáñez



# Ejemplos

---

## *Generación de números pseudoaleatorios*

Probemos con una secuencia de enteros generada aleatoriamente

Función `rand()` (`cstdlib`): entero aleatorio entre 0 y 32766

`srand()` (`cstdlib`): inicia la secuencia de números aleatorios

Acepta un entero que usa como semilla para iniciar la secuencia

¿Qué valor usar? Uno distinto en cada ejecución

→ El instante de tiempo actual (diferente cada vez)

Función `time()` (`ctime`): segundos transcurridos desde 1970

Requiere un argumento, que en nuestro caso será `NULL`

```
srand(time(NULL)); // Inicia la secuencia
```

```
...
```

```
numeros[0] = rand(); // Entre 0 y 32766
```

Luis Hernández Yáñez



## Cuenta de valores con $k$ dígitos

```
#include <iostream>
using namespace std;
#include <cstdlib> // srand() y rand()
#include <ctime> // time()

int digitos(int dato);

int main() {
    const int NUM = 100;
    typedef int tNum[NUM]; // Exactamente 100 números
    const int DIG = 5;
    typedef int tDig[DIG];
    tNum numeros;
    tDig numDig = { 0 }; // Inicializa todo el array a 0

    srand(time(NULL)); // Inicia la secuencia aleatoria
    ...
}
```



# Ejemplos

```
for (int i = 0; i < NUM; i++) { // Creamos la secuencia
    numeros[i] = rand(); // Entre 0 y 32766
}

for (int i = 0; i < NUM; i++) {
    // Recorremos la secuencia de enteros
    numDig[digitos(numeros[i]) - 1]++;
}

for (int i = 0; i < DIG; i++) {
    // Recorremos la secuencia de contadores
    cout << "De " << i + 1 << " díg. = " << numDig[i]
        << endl;
}
return 0;
}

int digitos(int dato) {
    ...
}
```



## Búsquedas en arrays



## Búsquedas en arrays

### *Esquema de búsqueda*

Inicialización

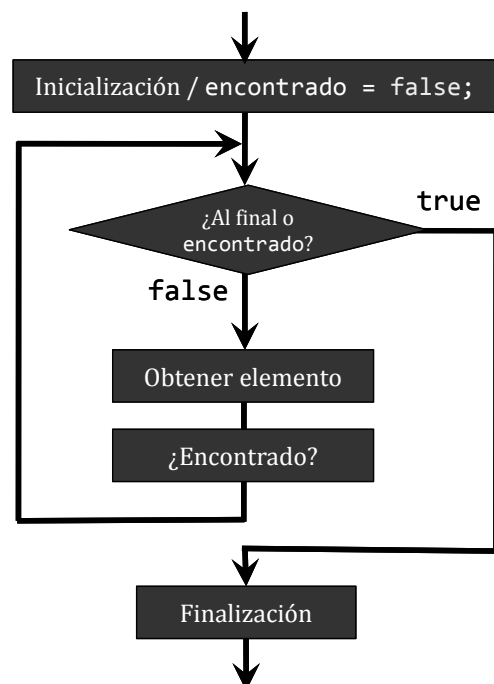
Mientras no se encuentre el elemento  
y no se esté al final de la secuencia:

Obtener el siguiente elemento

Comprobar si el elemento  
satisface la condición

Finalización

(tratar el elemento encontrado  
o indicar que no se ha encontrado)



# Búsquedas en arrays completos

Todas las posiciones ocupadas

```
int buscado;
bool encontrado = false;
cout << "Valor a buscar: ";
cin >> buscado;
int pos = 0;
while ((pos < N) && !encontrado) {
// Mientras no se llegue al final y no encontrado
    if (lista[pos] == buscado) {
        encontrado = true;
    }
    else {
        pos++;
    }
}
if (encontrado) // ...
```

```
const int N = 100;
typedef int tArray[N];
tArray lista;
```



# Búsquedas en arrays incompletos

*Con centinela*

```
int buscado;
cout << "Valor a buscar: ";
cin >> buscado;
int pos = 0;
bool encontrado = false;
while ((array[pos] != centinela) && !encontrado) {
    if (array[pos] == buscado) {
        encontrado = true;
    }
    else {
        pos++;
    }
}
if (encontrado) // ...
```

```
const int N = 10;
typedef int tArray[N];
tArray array;
const int centinela = -1;
```





# Búsquedas en arrays incompletos

## *Con contador*

```
int buscado;
cout << "Valor a buscar: ";
cin >> buscado;
int pos = 0;
bool encontrado = false;
while ((pos < miLista.contador)
      && !encontrado) {
    // Mientras no al final y no encontrado
    if (miLista.elementos[pos] == buscado) {
        encontrado = true;
    }
    else {
        pos++;
    }
}
if (encontrado) // ...
```

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elementos;
    int contador;
} tLista;
tLista miLista;
```



# Búsquedas por posición

## *Acceso directo a cualquier posición*

Acceso directo: *array[posición]*

Si se puede calcular la posición del elemento, su acceso es directo

```
typedef double tVentaMes[DIAS][SUCURSALES];
typedef struct {
    tVentaMes ventas;
    int dias;
} tMes;
typedef tMes tVentaAnual[MESES];
tVentaAnual anual;
```

*Ventas del cuarto día del tercer mes en la primera sucursal:*

```
anual[2].ventas[3][0]
```



## Ejemplo



## Primer valor por encima de un umbral

---

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

const int N = 100;
typedef double tArray[N];
typedef struct {
    tArray elementos;
    int contador;
} tLista;

void cargar(tLista &lista, bool &ok);

int main() {
    tLista lista;
    bool ok;
    cargar(lista, ok);
    if (!ok) {
        cout << "Error: no hay archivo o demasiados datos"
              << endl;
    }
}
```

umbral.cpp



# Primer valor por encima de un umbral

---

```
else {
    double umbral;
    cout << "Valor umbral: "; cin >> umbral;
    bool encontrado = false;
    int pos = 0;
    while ((pos < lista.contador) && !encontrado) {
        if (lista.elementos[pos] > umbral) {
            encontrado = true;
        }
        else {
            pos++;
        }
    }
    if (encontrado) {
        cout << "Valor en pos. " << pos + 1 << " ("
            << lista.elementos[pos] << ")" << endl;
    }
    else {
        cout << "¡No encontrado!" << endl;
    }
}
return 0;
}
```

Luis Hernández Yáñez



# Primer valor por encima de un umbral

---

```
void cargar(tLista &lista, bool &ok) {
    ifstream archivo;
    double dato;
    bool abierto = true, overflow = false;
    lista.contador = 0;
    archivo.open("datos.txt");
    if (!archivo.is_open()) {
        abierto = false;
    }
    else {
        archivo >> dato;
        while ((dato >= 0) && !overflow) {
            if (lista.contador == N) {
                overflow = true; // ¡Demasiados!
            }
            else {
                lista.elementos[lista.contador] = dato;
                lista.contador++;
                archivo >> dato;
            }
        }
        archivo.close();
    }
    ok = abierto && !overflow;
}
```

Luis Hernández Yáñez



## Recorridos y búsquedas en cadenas de caracteres



## Cadenas de caracteres

`inversa.cpp`

### *Recorridos y búsquedas en cadenas de caracteres*

Longitud de la cadena: `size()` o `length()`

Caso similar a los arrays con contador de elementos

Ejemplo: Recorrido de una cadena generando otra invertida

```
string cadena, inversa = "";
int pos;
char car;
// ... (lectura de cadena)
pos = 0;
while (pos < cadena.size()) {
    // Mientras no se llegue al final de la cadena
    car = cadena.at(pos);
    inversa = car + inversa; // Inserta car al principio
    pos++;
} // ...
```



## *Búsqueda de un carácter en una cadena*

```
string cadena;  
char buscado;  
int pos;  
bool encontrado;  
// ... (lectura de cadena)  
cout << "Introduce el carácter a buscar: ";  
cin >> buscado;  
pos = 0;  
encontrado = false;  
while ((pos < cadena.size()) && !encontrado) {  
    if (cadena.at(pos) == buscado) {  
        encontrado = true;  
    }  
    else {  
        pos++;  
    }  
}  
if (encontrado) // ...
```

Luis Hernández Yáñez



# Fundamentos de la programación

## Más ejemplos de manejo de arrays

Luis Hernández Yáñez



# Manejo de vectores

Tipo tVector para representar secuencias de N enteros:

```
const int N = 10;  
typedef int tVector[N];
```

Subprogramas:

- ✓ Dado un vector, mueve sus componentes un lugar a la derecha; el último componente se moverá al 1<sup>er</sup> lugar
- ✓ Dado un vector, calcula y devuelve la suma de los elementos que se encuentran en las posiciones pares del vector
- ✓ Dado un vector, encuentra y devuelve la componente mayor
- ✓ Dados dos vectores, devuelve un valor que indique si son iguales
- ✓ Dado un vector, determina si alguno de los valores almacenados en el vector es igual a la suma del resto de los valores del mismo; devuelve el índice del primero encontrado o -1 si no se encuentra
- ✓ Dado un vector, determina si alguno de los valores almacenados en el vector está repetido



# Manejo de vectores

vectores.cpp

```
void desplazar(tVector v) {  
    int aux = v[N - 1];  
  
    for (int i = N - 1; i > 0; i--) {  
        v[i] = v[i - 1];  
    }  
    v[0] = aux;  
}  
  
int sumaPares(const tVector v) {  
    int suma = 0;  
  
    for (int i = 0; i < N; i = i + 2) {  
        suma = suma + v[i];  
    }  
  
    return suma;  
}
```



# Manejo de vectores

---

```
int encuentraMayor(const tVector v) {
    int max = v[0], posMayor = 0;
    for (int i = 1; i < N; i++) {
        if (v[i] > max) {
            posMayor = i;
            max = v[i];
        }
    }
    return posMayor;
}

bool sonIguales(const tVector v1, const tVector v2) {
    //Implementación como búsqueda del primer elemento distinto
    bool encontrado = false;
    int i = 0;
    while ((i < N) && !encontrado) {
        encontrado = (v1[i] != v2[i]);
        i++;
    }
    return !encontrado;
}
```

Luis Hernández Yáñez



# Manejo de vectores

---

```
int compruebaSuma(const tVector v) {
    // ¿Alguno igual a la suma del resto?
    bool encontrado = false;
    int i = 0;
    int suma;
    while ((i < N) && !encontrado) {
        suma = 0;
        for (int j = 0; j < N; j++) {
            if (j != i) {
                suma = suma + v[j];
            }
        }
        encontrado = (suma == v[i]);
        i++;
    }
    if (!encontrado) {
        i = 0;
    }
    return i - 1;
}
```

Luis Hernández Yáñez



# Manejo de vectores

---

```
bool hayRepetidos(const tVector v) {
    bool encontrado = false;
    int i = 0, j;

    while ((i < N) && !encontrado) {
        j = i + 1;
        while ((j < N) && !encontrado) {
            encontrado = (v[i] == v[j]);
            j++;
        }
        i++;
    }

    return encontrado;
}
```



## Más vectores

---

Dado un vector de  $N$  caracteres  $v1$ , en el que no hay elementos repetidos, y otro vector de  $M$  caracteres  $v2$ , donde  $N \leq M$ , se quiere comprobar si todos los elementos del vector  $v1$  están también en el vector  $v2$

Por ejemplo, si:

$v1 =$  'a' 'h' 'i' 'm'

$v2 =$  'h' 'a' 'x' 'x' 'm' 'i'

El resultado sería cierto, ya que todos los elementos de  $v1$  están en  $v2$





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

const int N = 3;
const int M = 10;
typedef char tVector1[N];
typedef char tVector2[M];

bool esta(char dato, const tVector2 v2);
bool vectorIncluido(const tVector1 v1, const tVector2 v2);

int main() {
    tVector1 v1 = { 'a', 'b', 'c' };
    tVector2 v2 = { 'a', 'r', 'e', 't', 'z', 's', 'a', 'h', 'b', 'x' };
    bool ok = vectorIncluido(v1, v2);
    if (ok) {
        cout << "OK: v1 esta incluido en v2" << endl;
    }
    else {
        cout << "NO: v1 no esta incluido en v2" << endl;
    }
    return 0;
}
```

Luis Hernández Yáñez



# Manejo de vectores

```
bool esta(char dato, const tVector2 v2) {
    int i = 0;
    bool encontrado = (dato == v2[0]);

    while (!encontrado && (i < M - 1)) {
        i++;
        encontrado = (dato == v2[i]);
    }

    return encontrado;
}

bool vectorIncluido(const tVector1 v1, const tVector2 v2) {
    int i = 0;
    bool encontrado = esta(v1[0], v2);

    while (encontrado && (i < N - 1)) {
        i++;
        encontrado = esta(v1[i], v2);
    }

    return encontrado;
}
```

Luis Hernández Yáñez



# Anagramas

---

Un programa que lea dos cadenas del teclado y determine si una es un anagrama de la otra, es decir, si una cadena es una permutación de los caracteres de la otra.

Por ejemplo, "acre" es un anagrama de "cera" y de "arce". Ten en cuenta que puede haber letras repetidas ("carro", "llave").



# Anagramas

anagramas.cpp

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

int buscaCaracter(string cad, char c); // Índice o -1 si no está

int main() {
    string cad1, cad2;
    bool sonAnagramas = true;
    int numCar, posEnCad2;

    cout << "Introduce la primera cadena: ";
    getline(cin, cad1);
    cout << "Introduce la segunda cadena: ";
    getline(cin, cad2);
    if (cad1.length() != cad2.length()) { // No son anagramas
        sonAnagramas = false;
    }
    else {
        numCar = 0; // Contador de caracteres de la primera cadena
        while (sonAnagramas && (numCar < cad1.length())) {
            posEnCad2 = buscaCaracter(cad2, cad1.at(numCar));
```



# Anagramas

---

```
        if (posEnCad2 == -1) { //No se ha encontrado el caracter
            sonAnagramas = false;
        }
        else {
            cad2.erase(posEnCad2, 1);
        }
        numCar++;
    }
}

if (sonAnagramas) {
    cout << "Las palabras introducidas son anagramas" << endl;
}
else {
    cout << "Las palabras introducidas NO son anagramas" << endl;
}

return 0;
}
```



# Anagramas

---

```
int buscaCaracter(string cad, char c) {
    int pos = 0, lon = cad.length();
    bool encontrado = false;

    while ((pos < lon) && !encontrado) {
        if (cad.at(pos) == c) {
            encontrado = true;
        }
        else {
            pos++;
        }
    }
    if (!encontrado) {
        pos = -1;
    }

    return pos;
}
```



## Arrays multidimensionales



## Arrays multidimensionales

---

### *Arrays de varias dimensiones*

Varios tamaños en la declaración: cada uno con sus corchetes

```
typedef tipo_base nombre[tamaño1][tamaño2]...[tamañoN];
```

Varias dimensiones, tantas como tamaños se indiquen

```
typedef double tMatriz[50][100];  
tMatriz matriz;
```

Tabla bidimensional de 50 filas por 100 columnas:

	0	1	2	3	...	98	99
0					...		
1					...		
2					...		
...	...	...	...	...	...	...	...
48					...		
49					...		



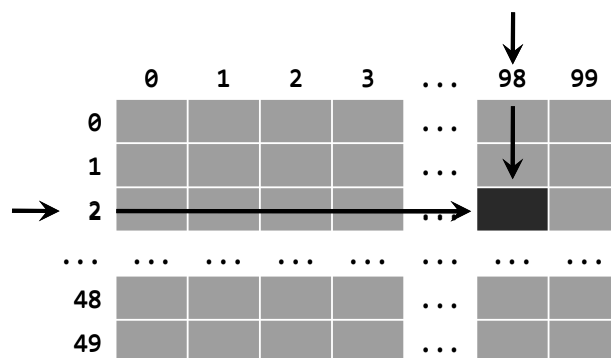
# Arrays multidimensionales

## *Arrays de varias dimensiones*

```
typedef double tMatriz[50][100];  
tMatriz matriz;
```

Cada elemento se localiza con dos índices, uno por dimensión

```
cout << matriz[2][98];
```



# Arrays multidimensionales



## *Arrays de varias dimensiones*

Podemos definir tantas dimensiones como necesitemos

```
typedef double tMatriz[5][10][20][10];  
tMatriz matriz;
```

Necesitaremos tantos índices como dimensiones:

```
cout << matriz[2][9][15][6];
```



# Arrays multidimensionales

## *Ejemplo de array bidimensional*

Temperaturas mínimas y máximas

Matriz bidimensional de días y mínima/máxima:

```
const int MaxDias = 31;
const int MED = 2; // N° de medidas
typedef double tTemp[MaxDias][MED]; // Día x mín./máx.
tTemp temp;
```

Ahora:

- ✓ `temp[i][0]` es la temperatura mínima del día `i+1`
- ✓ `temp[i][1]` es la temperatura máxima del día `i+1`



# Arrays multidimensionales

temp.cpp

```
int main() {
    const int MaxDias = 31;
    const int MED = 2; // N° de medidas
    typedef double tTemp[MaxDias][MED]; // Día x mín./máx.
    tTemp temp;
    double tMaxMedia = 0, tMinMedia = 0,
           tMaxAbs = -100, tMinAbs = 100;
    int dia = 0;
    double max, min;
    ifstream archivo;

    archivo.open("temp.txt");
    if (!archivo.is_open()) {
        cout << "No se ha podido abrir el archivo!" << endl;
    }
    else {
        archivo >> min >> max;
        // El archivo termina con -99 -99
        ...
    }
}
```



# Arrays multidimensionales

---

```
while (!(min == -99) && (max == -99)
        && (dia < MaxDias)) {
    temp[dia][0] = min;
    temp[dia][1] = max;
    dia++;
    archivo >> min >> max;
}
archivo.close();
for (int i = 0; i < dia; i++) {
    tMinMedia = tMinMedia + temp[i][0];
    if (temp[i][0] < tMinAbs) {
        tMinAbs = temp[i][0];
    }
    tMaxMedia = tMaxMedia + temp[i][1];
    if (temp[i][1] > tMaxAbs) {
        tMaxAbs = temp[i][1];
    }
}
...

```

Luis Hernández Yáñez



# Arrays multidimensionales

---

```
tMinMedia = tMinMedia / dia;
tMaxMedia = tMaxMedia / dia;
cout << "Temperaturas mínimas.-" << endl;
cout << "    Media = " << fixed << setprecision(1)
    << tMinMedia << " C    Mínima absoluta = "
    << setprecision(1) << tMinAbs << " C" << endl;
cout << "Temperaturas máximas.-" << endl;
cout << "    Media = " << fixed << setprecision(1)
    << tMaxMedia << " C    Máxima absoluta = "
    << setprecision(1) << tMaxAbs << " C" << endl;
}

return 0;
}

```

Luis Hernández Yáñez



# Inicialización de arrays multidimensionales

Podemos dar valores a los elementos de un array al declararlo

Arrays bidimensionales:

```
typedef int tArray[5][2];  
tArray cuads = {1,1, 2,4, 3,9, 4,16, 5,25};
```

Se asignan en el orden en el que los elementos están en memoria

La memoria es de una dimensión: secuencia de celdas

En memoria varían más rápidamente los índices de la derecha:

```
cuads[0][0] cuads[0][1] cuads[1][0] cuads[1][1] cuads[2][0]...
```

Para cada valor del primer índice: todos los valores del segundo



# Inicialización de arrays multidimensionales

## *Inicialización de un array bidimensional*

```
typedef int tArray[5][2];  
tArray cuads = {1,1, 2,4, 3,9, 4,16, 5,25};
```

	Memoria		0	1
cuads[0][0]	1	0	1	1
cuads[0][1]	1	1	2	4
cuads[1][0]	2	2	3	9
cuads[1][1]	4	3	4	16
cuads[2][0]	3	4	5	25
cuads[2][1]	9			
cuads[3][0]	4			
cuads[3][1]	16			
cuads[4][0]	5			
cuads[4][1]	25			



Si hay menos valores que elementos, el resto se inicializan a cero

Inicialización a cero de todo el array:

```
int cuads[5][2] = { 0 };
```





# Inicialización de arrays multidimensionales

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

	Memoria
matriz[0][0][0][0]	1
matriz[0][0][0][1]	2
matriz[0][0][0][2]	3
matriz[0][0][1][0]	4
matriz[0][0][1][1]	5
matriz[0][0][1][2]	6
matriz[0][1][0][0]	7
matriz[0][1][0][1]	8
matriz[0][1][0][2]	9
matriz[0][1][1][0]	10
matriz[0][1][1][1]	11
matriz[0][1][1][2]	12
matriz[0][2][0][0]	0
...	0



## Recorrido de un array bidimensional

```
const int FILAS = 10;  
const int COLUMNAS = 5;  
typedef double tMatriz[FILAS][COLUMNAS];  
tMatriz matriz;
```

Para cada *fila* (de 0 a FILAS – 1):

    Para cada *columna* (de 0 a COLUMNAS – 1):

        Procesar el elemento en *[fila][columna]*

```
for (int fila = 0; fila < FILAS; fila++) {  
    for (int columna = 0; columna < COLUMNAS; columna++) {  
        // Procesar matriz[fila][columna]  
    }  
}
```



# Ejemplo

## Ventas de todos los meses de un año

```
const int Meses = 12;
const int MaxDias = 31;
typedef double tVentas[Meses][MaxDias];
tVentas ventas; // Ventas de todo el año
typedef short int tDiasMes[Meses];
tDiasMes diasMes;
inicializa(diasMes); // N° de días de cada mes
// Pedimos las ventas de cada día del año...

for (int mes = 0; mes < Meses; mes++) {
    for (int dia = 0; dia < diasMes[mes]; dia++) {
        cout << "Ventas del día " << dia + 1
            << " del mes " << mes + 1 << ": ";
        cin >> ventas[mes][dia];
    }
}
```

Luis Hernández Yáñez



# Ejemplo

## Ventas de todos los meses de un año

		Días								
		0	1	2	3	4	...	28	29	30
Meses	0	201	125	234	112	156	...	234	543	667
	1	323	231	675	325	111	...			
	2	523	417	327	333	324	...	444	367	437
	3	145	845	654	212	562	...	354	548	
	4	327	652	555	222	777	...	428	999	666
	5	854	438	824	547	175	...	321	356	
	6	654	543	353	777	437	...	765	678	555
	7	327	541	164	563	327	...	538	159	235
	8	333	327	432	249	777	...	528	529	
	9	524	583	333	100	334	...	743	468	531
	10	217	427	585	218	843	...	777	555	
	11	222	666	512	400	259	...	438	637	879

Celdas no utilizadas

Luis Hernández Yáñez



# Recorrido de arrays N-dimensionales

```
const int DIM1 = 10;
const int DIM2 = 5;
const int DIM3 = 25;
const int DIM4 = 50;

typedef double tMatriz[DIM1][DIM2][DIM3][DIM4];

tMatriz matriz;

Bucles anidados, desde la primera dimensión hasta la última:
for (int n1 = 0; n1 < DIM1; n1++) {
    for (int n2 = 0; n2 < DIM2; n2++) {
        for (int n3 = 0; n3 < DIM3; n3++) {
            for (int n4 = 0; n4 < DIM4; n4++) {
                // Procesar matriz[n1][n2][n3][n4]
            }
        }
    }
}
```

Luis Hernández Yáñez



## Ejemplo

### *Ventas diarias de cuatro sucursales*

Cada mes del año: ingresos de cada sucursal cada día del mes  
Meses con distinto nº de días → junto con la matriz de ventas mensual guardamos el nº de días del mes concreto → estructura

```
const int DIAS = 31;
const int SUCURSALES = 4;
typedef double tVentaMes[DIAS][SUCURSALES];
typedef struct {
    tVentaMes ventas;
    int dias;
} tMes;
```

```
anual → tVentaAnual
anual[i] → tMes
anual[i].dias → int
anual[i].ventas → tVentaMes
anual[i].ventas[j][k] → double
```

```
const int MESES = 12;
typedef tMes tVentaAnual[MESES];
tVentaAnual anual;
```

Luis Hernández Yáñez



# Ejemplo

Cálculo de las ventas  
de todo el año:

Para cada mes...

Para cada día del mes...

Para cada sucursal...

Acumular las ventas

```
const int DIAS = 31;
const int SUCURSALES = 4;
typedef double
tVentaMes[DIAS][SUCURSALES];
typedef struct {
    tVentaMes ventas;
    int dias;
} tMes;

const int MESES = 12;
typedef tMes tVentaAnual[MESES];
tVentaAnual anual;
```

```
double total = 0;
for (int mes = 0; mes < MESES; mes++) {
    for (int dia = 0; dia < anual[mes].dias; dia++) {
        for (int suc = 0; suc < SUCURSALES; suc++) {
            total = total + anual[mes].ventas[dia][suc];
        }
    }
}
```

Luis Hernández Yáñez



# Búsqueda en un array multidimensional

```
bool encontrado = false;
int mes = 0, dia, suc;
while ((mes < MESES) && !encontrado) {
    dia = 0;
    while ((dia < anual[mes].dias) && !encontrado) {
        suc = 0;
        while ((suc < SUCURSALES) && !encontrado) {
            if (anual[mes].ventas[dia][suc] > umbral) {
                encontrado = true;
            }
            else {
                suc++;
            }
        }
        if (!encontrado) {
            dia++;
        }
    }
    if (!encontrado) {
        mes++;
    }
}
if (encontrado) { ...
```

Primer valor > umbral

Luis Hernández Yáñez






# Acerca de *Creative Commons*



## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.





## Algoritmos de ordenación

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Algoritmos de ordenación	651
Algoritmo de ordenación por inserción	654
Ordenación de arrays por inserción	665
Algoritmo de ordenación por inserción con intercambios	672
Claves de ordenación	680
Estabilidad de la ordenación	688
Complejidad y eficiencia	692
Ordenaciones naturales	694
Ordenación por selección directa	701
Método de la burbuja	716
Listas ordenadas	722
Búsquedas en listas ordenadas	729
Búsqueda binaria	731



## Algoritmos de ordenación



## Algoritmos de ordenación

---

### *Ordenación de listas*

array

125.40	76.95	328.80	254.62	435.00	164.29	316.05	219.99	93.45	756.62
0	1	2	3	4	5	6	7	8	9

Algoritmo de ordenación  
(de menor a mayor)



array

76.95	93.45	125.40	164.29	219.99	254.62	316.05	328.80	435.00	756.62
0	1	2	3	4	5	6	7	8	9

$\text{array}[i] \leq \text{array}[i + 1]$

Mostrar los datos en orden, facilitar las búsquedas, ...

Variadas formas de hacerlo (algoritmos)



# Algoritmos de ordenación

---

## *Ordenación de listas*

Los datos de la lista deben poderse comparar entre sí

Sentido de la ordenación:

- ✓ Ascendente (de menor a mayor)
- ✓ Descendente (de mayor a menor)

Algoritmos de ordenación básicos:

- ✓ Ordenación por *inserción*
- ✓ Ordenación por *selección directa*
- ✓ Ordenación por el *método de la burbuja*

Los algoritmos se basan en comparaciones e intercambios

Hay otros algoritmos de ordenación mejores



# Fundamentos de la programación

---

## Algoritmo de ordenación por inserción





# Ordenación por inserción

---

## *Algoritmo de ordenación por inserción*

Partimos de una lista vacía

Vamos insertando cada elemento en el lugar que le corresponda



Baraja de nueve cartas numeradas del 1 al 9

Las cartas están desordenadas

Ordenaremos de menor a mayor (ascendente)

Luis Hernández Yáñez



# Ordenación por inserción

---

## *Algoritmo de ordenación por inserción*



Colocamos el primer elemento en la lista vacía

Lista ordenada:



Luis Hernández Yáñez



# Ordenación por inserción

---

## Algoritmo de ordenación por inserción



El 7 es mayor que todos los elementos de la lista  
Lo insertamos al final

Lista ordenada:



# Ordenación por inserción

---

## Algoritmo de ordenación por inserción



Primer elemento (5) mayor que el nuevo (4):  
Desplazamos todos una posición a la derecha  
Insertamos el nuevo en la primera posición

*Hemos insertado el elemento en su lugar*

Lista ordenada:



# Ordenación por inserción

---

## *Algoritmo de ordenación por inserción*



9 es mayor que todos los elementos de la lista  
Lo insertamos al final

Lista ordenada:



# Ordenación por inserción

---

## *Algoritmo de ordenación por inserción*



Primer elemento (4) mayor que el nuevo (2):  
Desplazamos todos una posición a la derecha  
Insertamos el nuevo en la primera posición

Lista ordenada:



# Ordenación por inserción

---

## *Algoritmo de ordenación por inserción*



El 9 es el primer elemento mayor que el nuevo (8):  
Desplazamos desde ese hacia la derecha  
Insertamos donde estaba el 9

Lista ordenada:



# Ordenación por inserción

---

## *Algoritmo de ordenación por inserción*



Segundo elemento (4) mayor que el nuevo (3):  
Desplazamos desde ese hacia la derecha  
Insertamos donde estaba el 4

Lista ordenada:



# Ordenación por inserción

---

## *Algoritmo de ordenación por inserción*



Primer elemento (2) mayor que el nuevo (1):  
Desplazamos todos una posición a la derecha  
Insertamos el nuevo en la primera posición

Lista ordenada:



# Ordenación por inserción

---

## *Algoritmo de ordenación por inserción*



El 7 es el primer elemento mayor que el nuevo (6):  
Desplazamos desde ese hacia la derecha  
Insertamos donde estaba el 7

!!! LISTA ORDENADA !!!

Lista ordenada:



# Ordenación por inserción

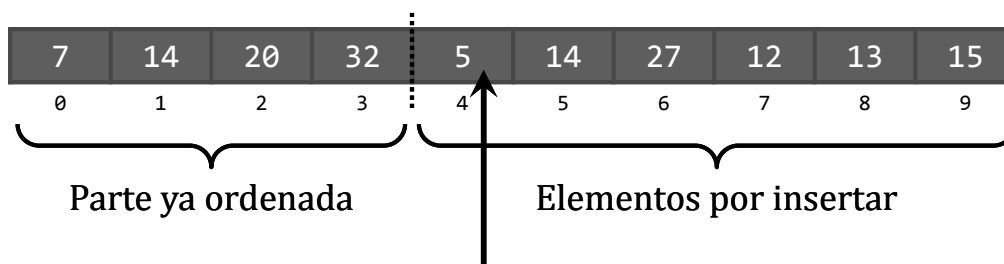
## Ordenación de arrays por inserción

El array contiene inicialmente la lista desordenada:

20	7	14	32	5	14	27	12	13	15
0	1	2	3	4	5	6	7	8	9

A medida que insertamos: dos zonas en el array

Parte ya ordenada y elementos por procesar



Luis Hernández Yáñez



# Ordenación por inserción

## Ordenación de arrays por inserción

Situación inicial: Lista ordenada con un solo elemento (primero)

20	7	14	32	5	14	27	12	13	15
0	1	2	3	4	5	6	7	8	9

Desde el segundo elemento del array hasta el último:  
Localizar el primer elemento mayor en lo ya ordenado

20	7	14	32	5	14	27	12	13	15
0	1	2	3	4	5	6	7	8	9



Primer elemento mayor o igual: índice 0

nuevo **7**

Luis Hernández Yáñez



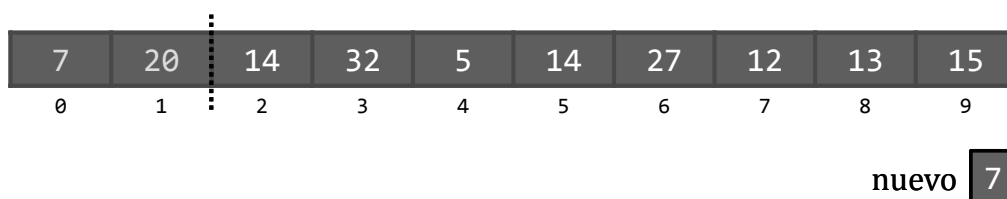
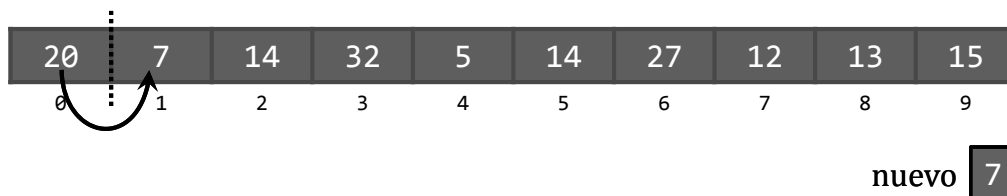
# Ordenación por inserción

## Ordenación de arrays por inserción

...

*Desplazar a la derecha los ordenados desde ese lugar*

*Insertar el nuevo en la posición que queda libre*



Luis Hernández Yáñez



# Ordenación de arrays por inserción

## Implementación

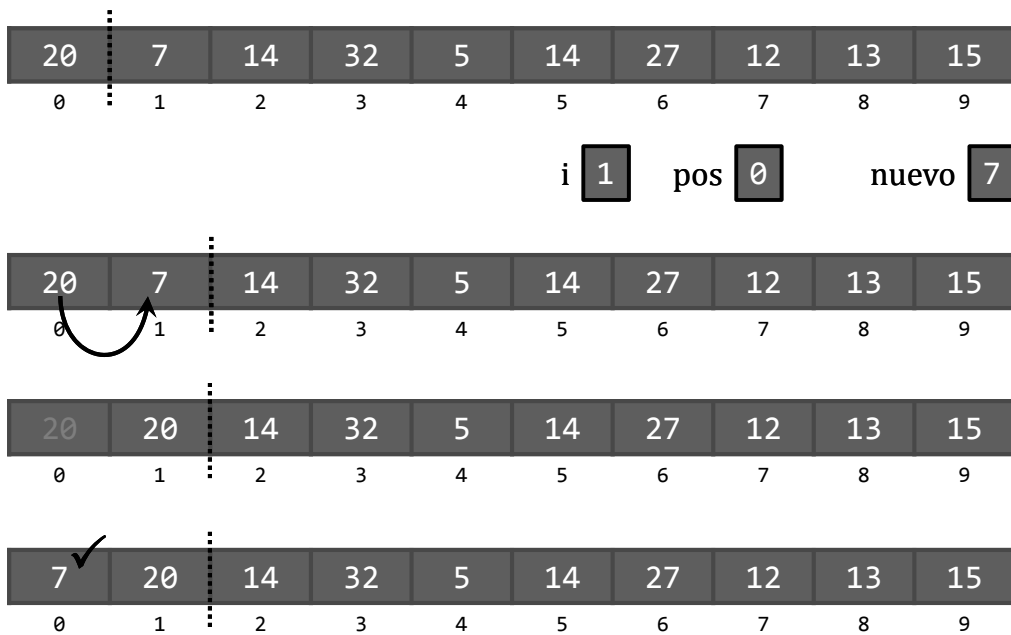
```
...
int nuevo, pos;
// Desde el segundo elemento hasta el último...
for (int i = 1; i < N; i++) {
    nuevo = lista[i];
    pos = 0;
    while ((pos < i) && !(lista[pos] > nuevo)) {
        pos++;
    }
    // pos: índice del primer mayor; i si no lo hay
    for (int j = i; j > pos; j--) {
        lista[j] = lista[j - 1];
    }
    lista[pos] = nuevo;
}
```

```
const int N = 15;
typedef int tLista[N];
tLista lista;
```

Luis Hernández Yáñez



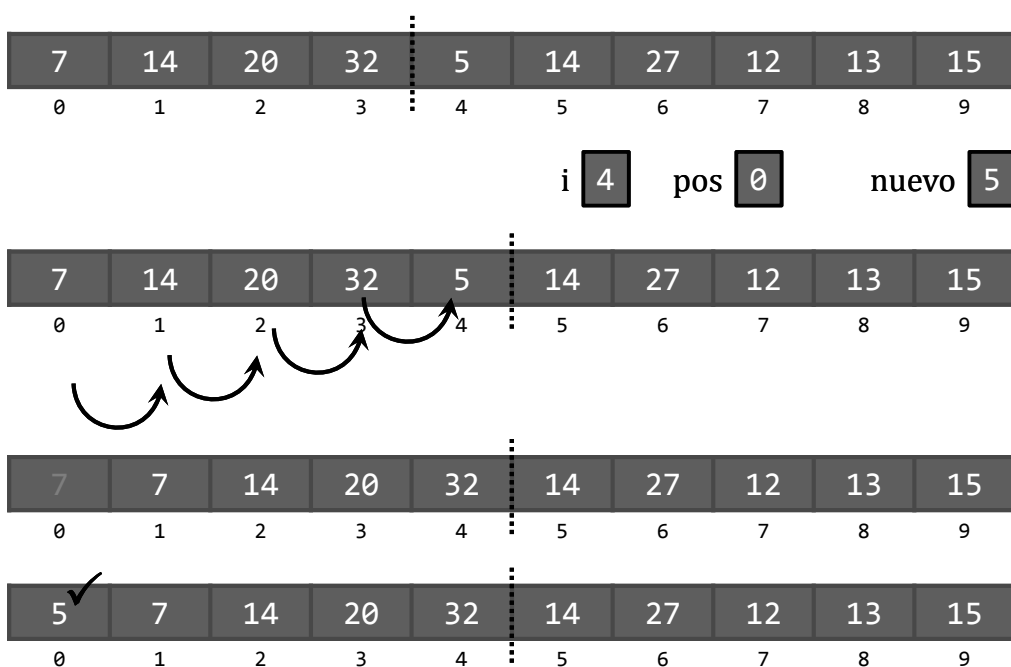
# Ordenación de arrays por inserción



Luis Hernández Yáñez



# Ordenación de arrays por inserción

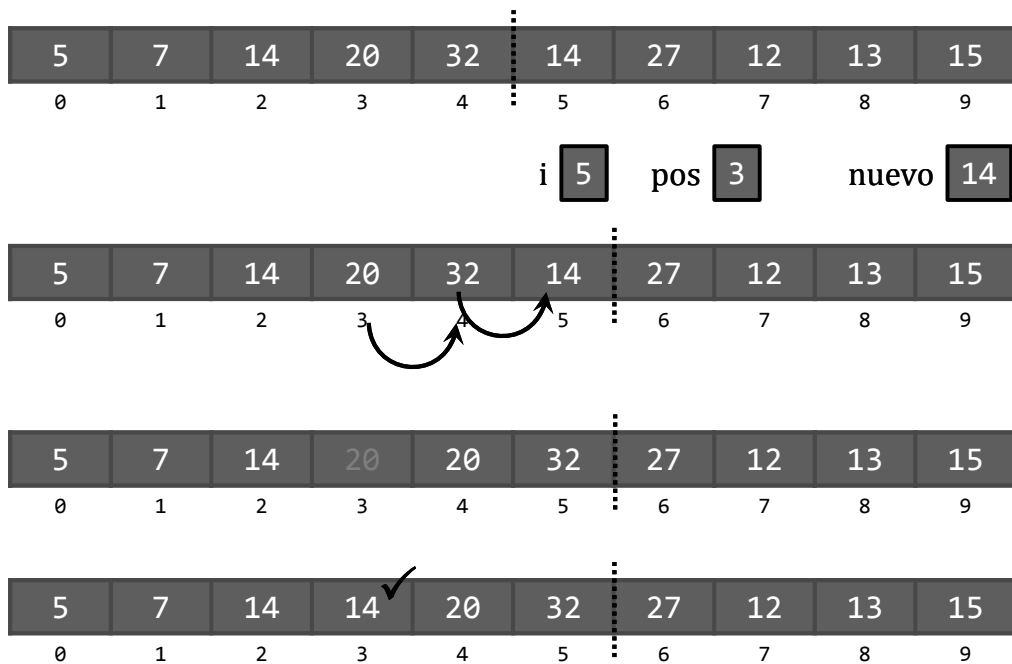


Luis Hernández Yáñez





# Ordenación de arrays por inserción



Luis Hernández Yáñez



# Fundamentos de la programación

## Algoritmo de ordenación por inserción con intercambios

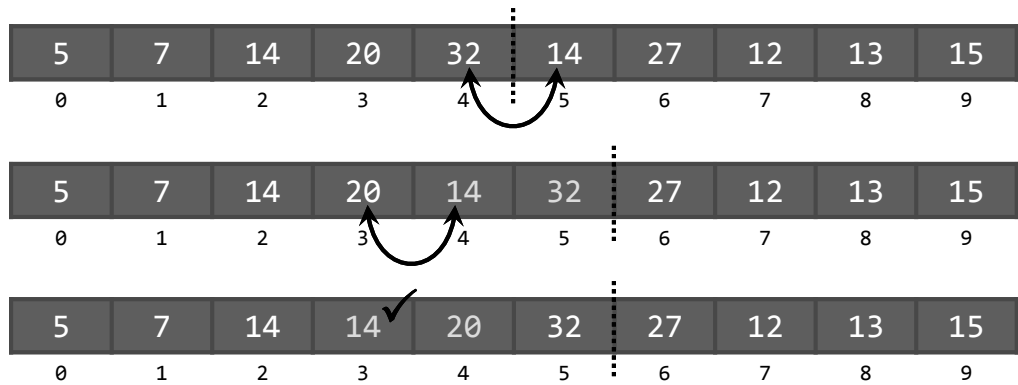
Luis Hernández Yáñez



# Ordenación por inserción con intercambios

La inserción de cada elemento se puede realizar con comparaciones e intercambios

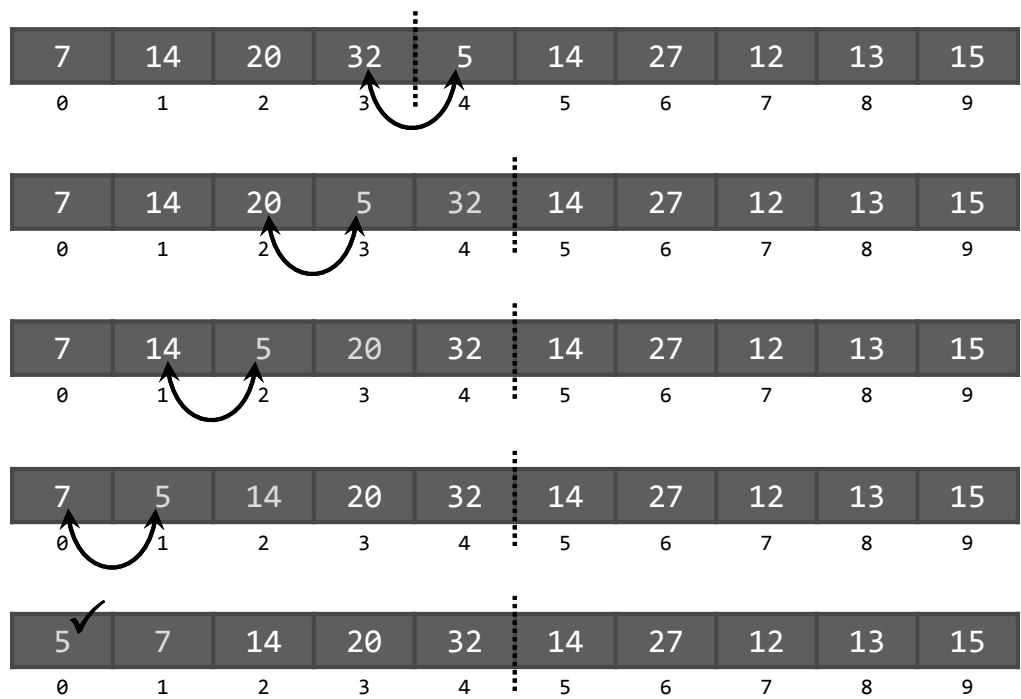
*Desde el segundo elemento hasta el último:  
Desde la posición del nuevo elemento a insertar:  
Mientras el anterior sea mayor, intercambiar*



Luis Hernández Yáñez



# Ordenación por inserción con intercambios



Luis Hernández Yáñez



# Ordenación por inserción con intercambios

```
const int N = 15;
typedef int tLista[N];
tLista lista;
```

```
...
int tmp, pos;
// Desde el segundo elemento hasta el último...
for (int i = 1; i < N; i++) {
    pos = i;
    // Mientras no al principio y anterior mayor...
    while ((pos > 0) && (lista[pos - 1] > lista[pos])) {
        // Intercambiar...
        tmp = lista[pos];
        lista[pos] = lista[pos - 1];
        lista[pos - 1] = tmp;
        pos--; // Posición anterior
    }
}
```

Luis Hernández Yáñez



# Ordenación por inserción con intercambios

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

const int N = 100;
typedef int tArray[N];
typedef struct { // Lista de longitud variable
    tArray elementos;
    int contador;
} tLista;

int main() {
    tLista lista;
    ifstream archivo;
    int dato, pos, tmp;
    lista.contador = 0;
    ...
}
```

insercion.cpp

Luis Hernández Yáñez



# Ordenación por inserción con intercambios

```
archivo.open("insercion.txt");
if (!archivo.is_open()) {
    cout << "Error de apertura de archivo!" << endl;
}
else {
    archivo >> dato;
    while ((lista.contador < N) && (dato != -1)) {
        // Centinela -1 al final
        lista.elementos[lista.contador] = dato;
        lista.contador++;
        archivo >> dato;
    }
    archivo.close();
    // Si hay más de N ignoramos el resto
    cout << "Antes de ordenar:" << endl;
    for (int i = 0; i < lista.contador; i++) {
        cout << lista.elementos[i] << " ";
    }
    cout << endl;    ...
}
```



# Ordenación por inserción con intercambios

```
for (int i = 1; i < lista.contador; i++) {
    pos = i;
    while ((pos > 0)
        && (lista.elementos[pos-1] > lista.elementos[pos]))
    {
        tmp = lista.elementos[pos];
        lista.elementos[pos] = lista.elementos[pos - 1];
        lista.elementos[pos - 1] = tmp;
        pos--;
    }
}
cout << "Después de ordenar:" << endl;
for (int i = 0; i < lista.contador; i++) {
    cout << lista.elementos[i] << " ";
}
cout << endl;
}
return 0;
}
```



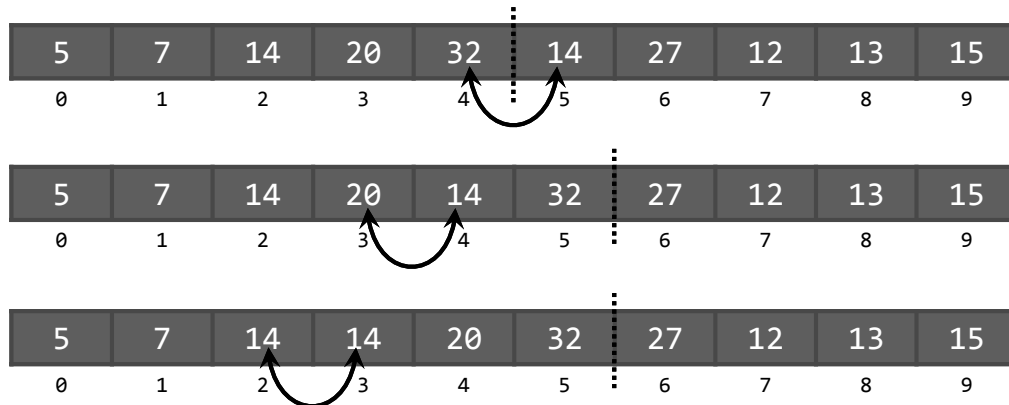
# Ordenación por inserción con intercambios

## Consideración de implementación

¿Operador relacional adecuado?

`lista[pos - 1] > lista[pos]` ?

Con `>=` se realizan intercambios inútiles:



Luis Hernández Yáñez



# Fundamentos de la programación

## Claves de ordenación

Luis Hernández Yáñez



# Ordenación por inserción

---

## Claves de ordenación

Elementos que son estructuras con varios campos:

```
const int N = 15;
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tDato;
typedef tDato tLista[N];
tLista lista;
```

Clave de ordenación:

Campo en el que se basan las comparaciones



# Ordenación por inserción

---

## Claves de ordenación

```
tDato tmp;
while ((pos > 0)
    && (lista[pos - 1].nombre > lista[pos].nombre)) {
    tmp = lista[pos];
    lista[pos] = lista[pos - 1];
    lista[pos - 1] = tmp;
    pos--;
}
```

Comparación: campo concreto

Intercambio: elementos completos



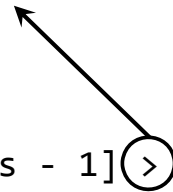
# Ordenación por inserción

## Claves de ordenación

Función para la comparación:

```
bool operator>(tDato opIzq, tDato opDer) {  
    return (opIzq.nombre > opDer.nombre);  
}
```

```
tDato tmp;  
while ((pos > 0) && (lista[pos - 1] > lista[pos])) {  
    tmp = lista[pos];  
    lista[pos] = lista[pos - 1];  
    lista[pos - 1] = tmp;  
    pos--;  
}
```



# Ordenación por inserción

claves.cpp

## Claves de ordenación

```
#include <iostream>  
#include <string>  
using namespace std;  
#include <fstream>  
#include <iomanip>  
const int N = 15;  
typedef struct {  
    int codigo;  
    string nombre;  
    double sueldo;  
} tDato;  
typedef tDato tArray[N];  
typedef struct {  
    tArray datos;  
    int cont;  
} tLista;  
...
```



# Ordenación por inserción

---

```
void mostrar(tLista lista);
bool operator>(tDato opIzq, tDato opDer);

int main() {
    tLista lista;
    ifstream archivo;
    lista.cont = 0;
    archivo.open("datos.txt");
    if (!archivo.is_open()) {
        cout << "Error de apertura del archivo!" << endl;
    }
    else {
        tDato dato;
        archivo >> dato.codigo;
        while ((lista.cont < N) && (dato.codigo != -1)) {
            archivo >> dato.nombre >> dato.sueldo;
            lista.datos[lista.cont] = dato;
            lista.cont++;
            archivo >> dato.codigo;
        }
        archivo.close();          ...
    }
}
```

Luis Hernández Yáñez



# Ordenación por inserción

---

```
cout << "Antes de ordenar:" << endl;
mostrar(lista);
for (int i = 1; i < lista.cont; i++) {
    // Desde el segundo elemento hasta el último
    int pos = i;
    while ((pos > 0)
        && (lista.datos[pos-1] > lista.datos[pos])) {
        tDato tmp;
        tmp = lista.datos[pos];
        lista.datos[pos] = lista.datos[pos - 1];
        lista.datos[pos - 1] = tmp;
        pos--;
    }
}
cout << "Después de ordenar:" << endl;
mostrar(lista);
}
return 0;
}
```

Luis Hernández Yáñez





# Ordenación por inserción

```
void mostrar(tLista lista) {
    for (int i = 0; i < lista.cont; i++) {
        cout << setw(10)
            << lista.datos[i].codigo
            << setw(20)
            << lista.datos[i].nombre
            << setw(12)
            << fixed
            << setprecision(2)
            << lista.datos[i].sueldo
            << endl;
    }
}

bool operator>(tDato opIzq, tDato opDer) {
    return (opIzq.nombre > opDer.nombre);
}
```

```
Antes de ordenar:
10000 Sergei 100000.00
10000 Hernandez 150000.00
11111 Benitez 100000.00
11111 Urpiano 90000.00
11111 Perez 90000.00
11111 Duran 120000.00
12345 Alvarez 120000.00
12345 Gomez 100000.00
12345 Sanchez 90000.00
12345 Turégano 100000.00
21112 Domínguez 90000.00
21112 Jiménez 100000.00
22222 Fernández 120000.00
33333 Tarazona 120000.00

Después de ordenar:
11111 Benitez 100000.00
21112 Domínguez 90000.00
11111 Duran 120000.00
22222 Fernández 120000.00
12345 Gomez 100000.00
10000 Hernandez 150000.00
21112 Jiménez 100000.00
11111 Perez 90000.00
10000 Sergei 100000.00
12345 Sanchez 90000.00
33333 Tarazona 120000.00
12345 Turégano 100000.00
11111 Urpiano 90000.00
12345 Alvarez 120000.00
```

Cambia a codigo o sueldo para ordenar por otros campos



# Fundamentos de la programación

## Estabilidad de la ordenación



# Estabilidad de la ordenación

## Algoritmos de ordenación estables

Al ordenar por otra clave una lista ya ordenada, la segunda ordenación preserva el orden de la primera

tDato: tres posibles claves de ordenación (campos)

Codigo  
Nombre  
Sueldo

12345	Álvarez	120000
11111	Benítez	100000
21112	Domínguez	90000
11111	Durán	120000
22222	Fernández	120000
12345	Gómez	100000
10000	Hernández	150000
21112	Jiménez	100000
11111	Pérez	90000
12345	Sánchez	90000
10000	Sergei	100000
33333	Tarazona	120000
12345	Turégano	100000
11111	Urpiano	90000

Lista ordenada por Nombre →



# Estabilidad de la ordenación

Ordenamos ahora por el campo Codigo:

10000	Sergei	100000
10000	Hernández	150000
11111	Urpiano	90000
11111	Benítez	100000
11111	Pérez	90000
11111	Durán	120000
12345	Sánchez	90000
12345	Álvarez	120000
12345	Turégano	100000
12345	Gómez	100000
21112	Domínguez	90000
21112	Jiménez	100000
22222	Fernández	120000
33333	Tarazona	120000

No estable:  
Los nombres no mantienen  
sus posiciones relativas

10000	Hernández	150000
10000	Sergei	100000
11111	Benítez	100000
11111	Durán	120000
11111	Pérez	90000
11111	Urpiano	90000
12345	Álvarez	120000
12345	Gómez	100000
12345	Sánchez	90000
12345	Turégano	100000
21112	Domínguez	90000
21112	Jiménez	100000
22222	Fernández	120000
33333	Tarazona	120000

Estable:  
Los nombres mantienen  
sus posiciones relativas



# Estabilidad de la ordenación

## Ordenación por inserción

Estable siempre que utilicemos  $< o >$  Con  $\leq o \geq$  no es estable

Ordenamos por sueldo:

A igual sueldo, ordenado por códigos y a igual código, por nombres

10000	Hernández	150000	11111	Pérez	90000
10000	Sergei	100000	11111	Urpiano	90000
11111	Benítez	100000	12345	Sánchez	90000
11111	Durán	120000	21112	Domínguez	90000
11111	Pérez	90000	10000	Sergei	100000
11111	Urpiano	90000	11111	Benítez	100000
12345	Álvarez	120000	12345	Gómez	100000
12345	Gómez	100000	12345	Turégano	100000
12345	Sánchez	90000	21112	Jiménez	100000
12345	Turégano	100000	11111	Durán	120000
21112	Domínguez	90000	12345	Álvarez	120000
21112	Jiménez	100000	22222	Fernández	120000
22222	Fernández	120000	33333	Tarazona	120000
33333	Tarazona	120000	10000	Hernández	150000

Luis Hernández Yáñez



# Fundamentos de la programación

## Complejidad y eficiencia

Luis Hernández Yáñez



# Complejidad y eficiencia

## Casos de estudio para los algoritmos de ordenación

- ✓ Lista inicialmente ordenada

5	7	12	13	14	14	15	20	27	32
0	1	2	3	4	5	6	7	8	9

- ✓ Lista inicialmente ordenada al revés

32	27	20	15	14	14	13	12	7	5
0	1	2	3	4	5	6	7	8	9

- ✓ Lista con disposición inicial aleatoria

13	20	7	14	12	32	27	14	5	15
0	1	2	3	4	5	6	7	8	9

¿Trabaja menos, más o igual la ordenación en cada caso?



# Complejidad y eficiencia

## Ordenaciones naturales

Si el algoritmo trabaja menos cuanto *más ordenada* está inicialmente la lista, se dice que la ordenación es *natural*

Ordenación por inserción con la lista inicialmente ordenada:

- ✓ Versión que busca el lugar primero y luego desplaza:  
No hay desplazamientos; mismo número de comparaciones  
Comportamiento no natural
- ✓ Versión con intercambios:  
Trabaja mucho menos; basta una comparación cada vez  
Comportamiento natural



# Complejidad y eficiencia

## *Elección de un algoritmo de ordenación*

¿Cómo de bueno es cada algoritmo?

¿Cuánto tarda en comparación con otros algoritmos?

Algoritmos más eficientes: los de menor complejidad

Tardan menos en realizar la misma tarea

Comparamos en orden de complejidad:  $O()$

En función de la dimensión de la lista a ordenar:  $N$

$O() = f(N)$

Operaciones que realiza el algoritmo de ordenación:

✓ Comparaciones

✓ Intercambios

Asumimos que tardan un tiempo similar



# Complejidad y eficiencia

## *Cálculo de la complejidad*

Ordenación por inserción (con intercambios):

```
...
for (int i = 1; i < N; i++) {
    int pos = i;
    while ((pos > 0) && (lista[pos - 1] > lista[pos])) {
        int tmp;
        tmp = lista[pos];
        lista[pos] = lista[pos - 1];
        lista[pos - 1] = tmp;
        pos--;
    }
}
```

Comparación

Intercambio

Intercambios y comparaciones:

Tantos como ciclos realicen los correspondientes bucles



# Complejidad y eficiencia

## Cálculo de la complejidad

```
... N - 1 ciclos
for (int i = 1; i < N; i++) {
    int pos = i; N° variable de ciclos
    while ((pos > 0) && (lista[pos - 1] > lista[pos])) {
        int tmp;
        tmp = lista[pos];
        lista[pos] = lista[pos - 1];
        lista[pos - 1] = tmp;
        pos--;
    }
}
```

Caso en el que el while se ejecuta más: *caso peor*

Caso en el que se ejecuta menos: *caso mejor*

Luis Hernández Yáñez



# Complejidad y eficiencia

## Cálculo de la complejidad

- ✓ Caso mejor: lista inicialmente ordenada  
La primera comparación falla: ningún intercambio  
 $(N - 1) * (1 \text{ comparación} + 0 \text{ intercambios}) = N - 1 \rightarrow O(N)$
- ✓ Caso peor: lista inicialmente ordenada al revés  
Para cada pos, entre i y 1: 1 comparación y 1 intercambio  
 $1 + 2 + 3 + 4 + \dots + (N - 1)$   
 $((N - 1) + 1) * (N - 1) / 2$   
 $N * (N - 1) / 2$   
 $(N^2 - N) / 2 \rightarrow O(N^2)$

Notación *O grande*: orden de complejidad en base a N

El término en N que más rápidamente crece al crecer N

En el caso peor,  $N^2$  crece más rápido que N  $\rightarrow O(N^2)$

(Ignoramos las constantes, como 2)

Luis Hernández Yáñez



# Complejidad y eficiencia

---

## *Ordenación por inserción (con intercambios)*

✓ Caso mejor:  $O(N)$

✓ Caso peor:  $O(N^2)$

Caso medio (distribución aleatoria de los elementos):  $O(N^2)$

Hay algoritmos de ordenación mejores



# Complejidad y eficiencia

---

## *Órdenes de complejidad*

$O(\log N) < O(N) < O(N \log N) < O(N^2) < O(N^3) \dots$

N	$\log_2 N$	$N^2$
1	0	1
2	1	4
4	2	16
8	3	64
16	4	256
32	5	1024
64	6	4096
128	7	16384
256	8	65536
...		



## Ordenación por selección directa



## Ordenación por selección directa

---

### *Algoritmo de ordenación por selección directa*

Seleccionar el siguiente elemento menor de los que quedan



Lista ordenada:

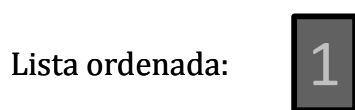




# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

Seleccionar el siguiente elemento menor de los que quedan



Luis Hernández Yáñez



# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

Seleccionar el siguiente elemento menor de los que quedan



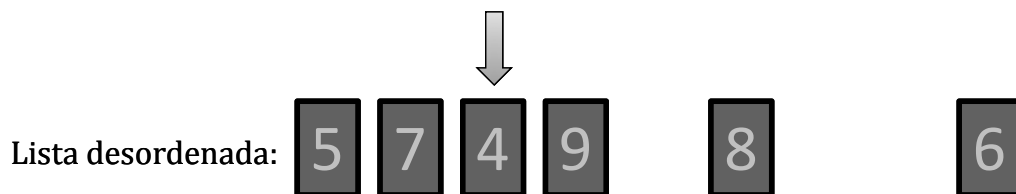
Luis Hernández Yáñez



# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

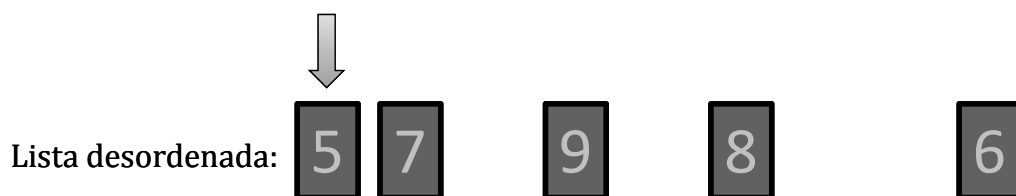
Seleccionar el siguiente elemento menor de los que quedan



# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

Seleccionar el siguiente elemento menor de los que quedan



# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

Seleccionar el siguiente elemento menor de los que quedan

Lista desordenada:

7

9

8

6



Lista ordenada:

1

2

3

4

5



# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

Seleccionar el siguiente elemento menor de los que quedan

Lista desordenada:

7

9

8



Lista ordenada:

1

2

3

4

5

6

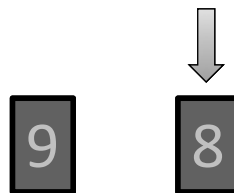


# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

Seleccionar el siguiente elemento menor de los que quedan

Lista desordenada:



Lista ordenada:



# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

Seleccionar el siguiente elemento menor de los que quedan

Lista desordenada:



Lista ordenada:



# Ordenación por selección directa

## Algoritmo de ordenación por selección directa

Seleccionar el siguiente elemento menor de los que quedan

Lista desordenada:

!!! LISTA ORDENADA !!!

Lista ordenada:



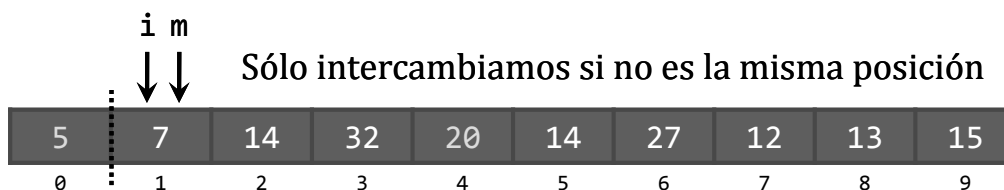
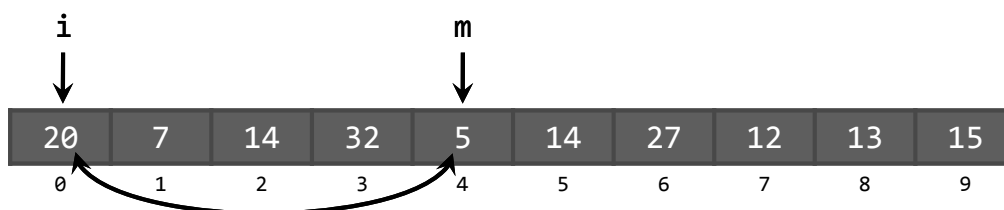
Luis Hernández Yáñez



# Ordenación por selección directa

## Ordenación de un array por selección directa

Desde el primer elemento ( $i = 0$ ) hasta el penúltimo ( $N-2$ ):  
Menor elemento (en  $m$ ) entre  $i + 1$  y el último ( $N-1$ )  
Intercambiar los elementos en  $i$  y  $m$  si no son el mismo

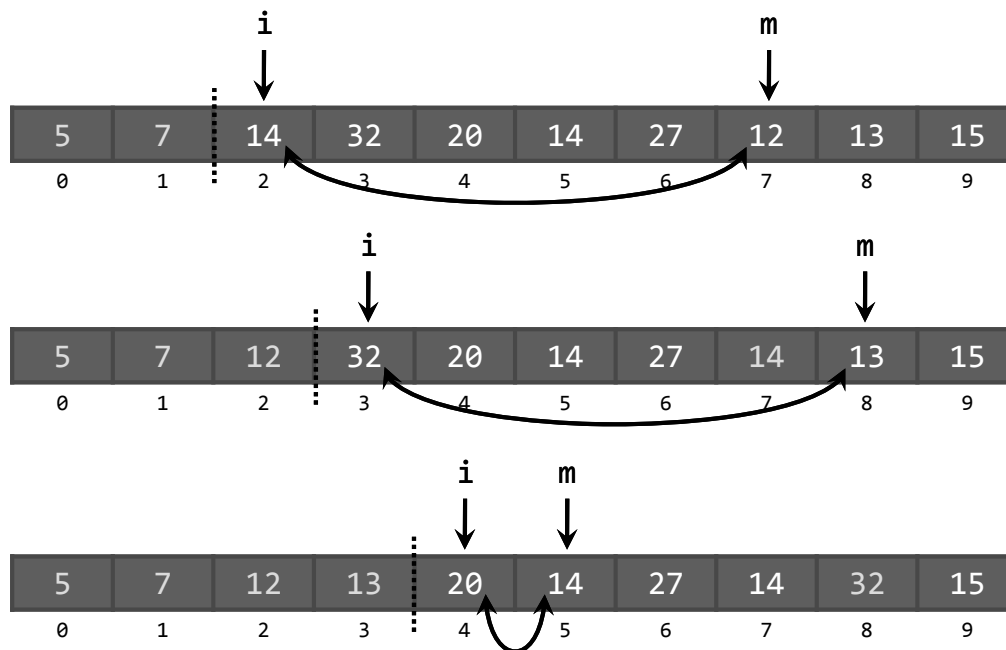


Luis Hernández Yáñez



# Ordenación por selección directa

## Ordenación de un array por selección directa



Luis Hernández Yáñez



# Ordenación por selección directa

seleccion.cpp

## Implementación

```
const int N = 15;
typedef int tLista[N];
tLista lista;
```

```
// Desde el primer elemento hasta el penúltimo...
for (int i = 0; i < N - 1; i++) {
    int menor = i;
    // Desde i + 1 hasta el final...
    for (int j = i + 1; j < N; j++) {
        if (lista[j] < lista[menor]) {
            menor = j;
        }
    }
    if (menor > i) {
        int tmp;
        tmp = lista[i];
        lista[i] = lista[menor];
        lista[menor] = tmp;
    }
}
```

Luis Hernández Yáñez



# Ordenación por selección directa

---

## *Complejidad de la ordenación por selección directa*

*¿Cuántas comparaciones se realizan?*

Bucle externo:  $N - 1$  ciclos

Tantas comparaciones como elementos queden en la lista:

$$(N - 1) + (N - 2) + (N - 3) + \dots + 3 + 2 + 1 =$$

$$N \times (N - 1) / 2 = (N^2 - N) / 2 \rightarrow O(N^2)$$

Mismo número de comparaciones en todos los casos

Complejidad:  $O(N^2)$  Igual que el método de inserción

Algo mejor (menos intercambios; uno en cada paso)

No es estable: intercambios “a larga distancia”

No se garantiza que se mantenga el mismo orden relativo original

Comportamiento no natural (trabaja siempre lo mismo)



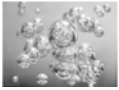
---

# Fundamentos de la programación

## Método de la burbuja



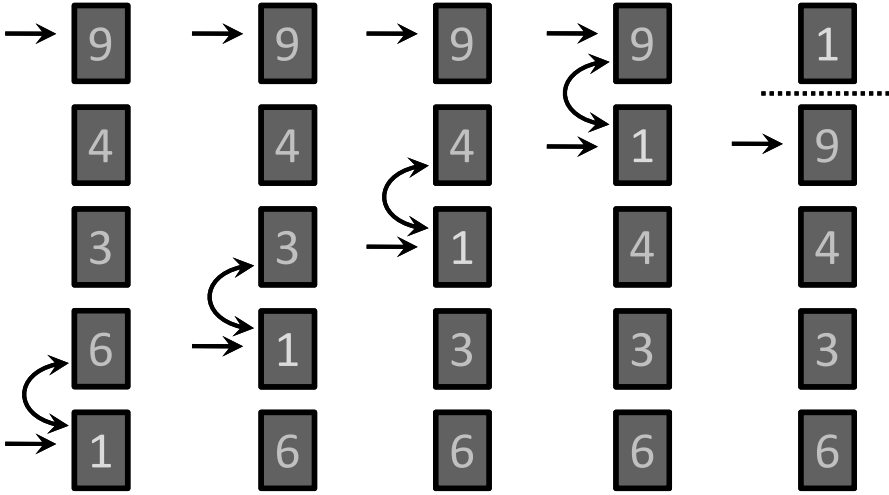
# Método de la burbuja



## Algoritmo de ordenación por el método de la burbuja

Variación del método de selección directa

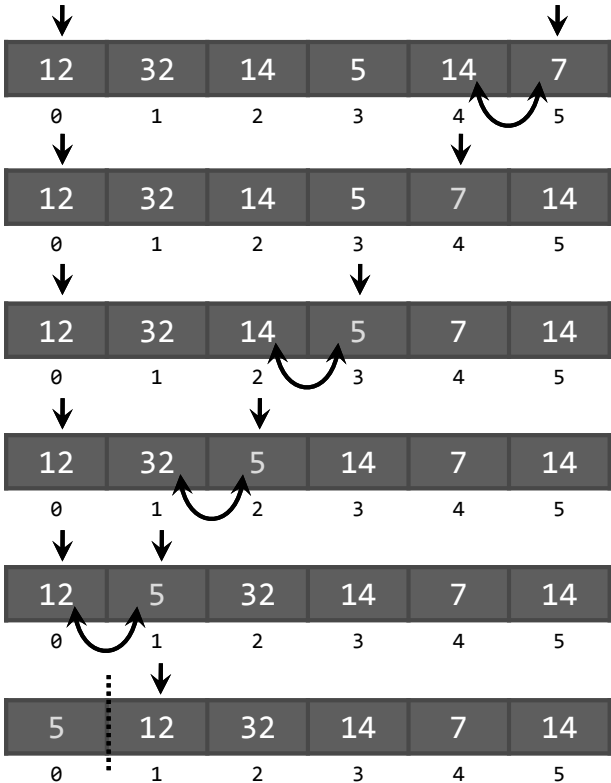
El elemento menor va *ascendiendo* hasta alcanzar su posición



Luis Hernández Yáñez



# Método de la burbuja



Luis Hernández Yáñez





## Ordenación de un array por el método de la burbuja

Desde el primero ( $i = 0$ ), hasta el penúltimo ( $N - 2$ ):  
Desde el último ( $j = N - 1$ ), hasta  $i + 1$ :  
Si elemento en  $j <$  elemento en  $j - 1$ , intercambiarlos

```
...
int tmp;
// Del primero al penúltimo...
for (int i = 0; i < N - 1; i++) {
    // Desde el último hasta el siguiente a i...
    for (int j = N - 1; j > i; j--) {
        if (lista[j] < lista[j - 1]) {
            tmp = lista[j];
            lista[j] = lista[j - 1];
            lista[j - 1] = tmp;
        }
    }
}
```

```
const int N = 10;
typedef int tLista[N];
tLista lista;
```

Luis Hernández Yáñez



# Método de la burbuja

## Algoritmo de ordenación por el método de la burbuja

Complejidad:  $O(N^2)$

Comportamiento no natural

Estable (mantiene el orden relativo)

Mejora:

Si en un paso del bucle exterior no ha habido intercambios:

La lista ya está ordenada (no es necesario seguir)

14	14	14	12
16	16	12	14
35	12	16	16
12	35	35	35
50	50	50	50

La lista ya está ordenada  
No hace falta seguir

Luis Hernández Yáñez



```
bool inter = true;
int i = 0;
// Desde el 1º hasta el penúltimo si hay intercambios...
while ((i < N - 1) && inter) {
    inter = false;
    // Desde el último hasta el siguiente a i...
    for (int j = N - 1; j > i; j--) {
        if (lista[j] < lista[j - 1]) {
            int tmp;
            tmp = lista[j];
            lista[j] = lista[j - 1];
            lista[j - 1] = tmp;
            inter = true;
        }
    }
    if (inter) {
        i++;
    }
}
```

Esta variación sí tiene un comportamiento natural



## Fundamentos de la programación

# Listas ordenadas



# Listas ordenadas

---

## *Gestión de listas ordenadas*

Casi todas las tareas se realizan igual que en listas sin orden

Operaciones que tengan en cuenta el orden:

- ✓ Inserción de un nuevo elemento: debe seguir en orden
- ✓ Búsquedas más eficientes

¿Y la carga desde archivo?

- ✓ Si los elementos se guardaron en orden: se lee igual
- ✓ Si los elementos no están ordenados en el archivo: insertar



# Gestión de listas ordenadas

lista.cpp

Declaraciones: Iguales que para listas sin orden

```
const int N = 20;
```

```
typedef struct {  
    int codigo;  
    string nombre;  
    double sueldo;  
} tRegistro;
```

```
typedef tRegistro tArray[N];
```

```
typedef struct {  
    tArray registros;  
    int cont;  
} tLista;
```



# Gestión de listas ordenadas

---

Subprogramas: Misma declaración que para listas sin orden

```
void mostrarDato(int pos, tRegistro registro);
void mostrar(tLista lista);
bool operator>(tRegistro opIzq, tRegistro opDer);
bool operator<(tRegistro opIzq, tRegistro opDer);
tRegistro nuevo();
void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N
int buscar(tLista lista, string nombre);
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
```



# Gestión de listas ordenadas

---

Nuevas implementaciones:

- ✓ Operadores relacionales
- ✓ Inserción (mantener el orden)
- ✓ Búsqueda (más eficiente)

Se guarda la lista en orden, por lo que cargar() no cambia

```
bool operator>(tRegistro opIzq, tRegistro opDer) {
    return opIzq.nombre > opDer.nombre;
}
bool operator<(tRegistro opIzq, tRegistro opDer) {
    return opIzq.nombre < opDer.nombre;
}
```



# Gestión de listas ordenadas

---

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        }
        // Insertamos en la posición i (primer mayor o igual)
        for (int j = lista.cont; j > i; j--) {
            // Desplazamos una posición a la derecha
            lista.registros[j] = lista.registros[j - 1];
        }
        lista.registros[i] = registro;
        lista.cont++;
    }
}
```

Luis Hernández Yáñez



# Fundamentos de la programación

---

## Búsquedas en listas ordenadas

Luis Hernández Yáñez



# Búsquedas en listas ordenadas

## *Búsqueda de un elemento en una secuencia*

No ordenada: recorremos hasta encontrarlo o al final

Ordenada: recorremos hasta encontrarlo o mayor / al final

5	7	12	13	14	14	15	20	27	32
0	1	2	3	4	5	6	7	8	9

Buscamos el 36: al llegar al final sabemos que no está

Buscamos el 17: al llegar al 20 ya sabemos que no está

Condiciones de terminación:

- ✓ Se llega al final
- ✓ Se encuentra el elemento buscado
- ✓ Se encuentra uno mayor
- Mientras no al final y el valor sea menor que el buscado



# Búsquedas en listas ordenadas

```
int buscado;
cout << "Valor a buscar: ";
cin >> buscado;
int i = 0;
while ((i < N) && (lista[i] < buscado)) {
    i++;
}
// Ahora, o estamos al final o lista[i] >= buscado
if (i == N) { // Al final: no se ha encontrado
    cout << "No encontrado!" << endl;
}
else if (lista[i] == buscado) { // Encontrado!
    cout << "Encontrado en posición " << i + 1 << endl;
}
else { // Hemos encontrado uno mayor
    cout << "No encontrado!" << endl;
}
```

```
const int N = 10;
typedef int tLista[N];
tLista lista;
```

Complejidad:  $O(N)$



## Búsqueda binaria

Luis Hernández Yáñez



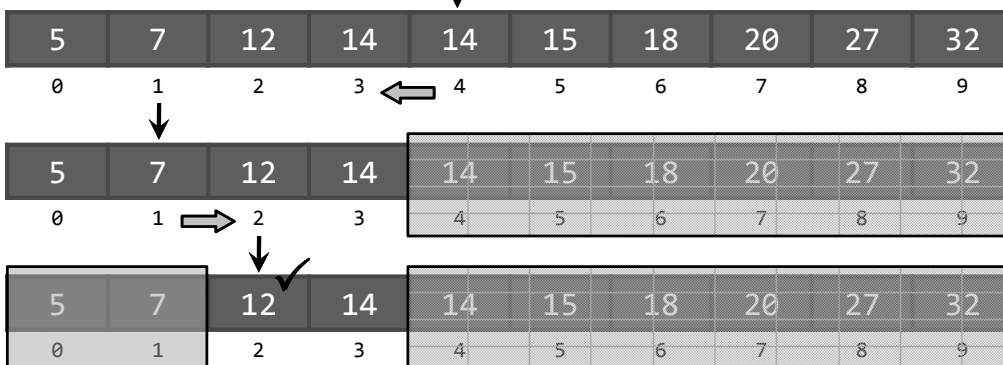
## Búsqueda binaria

Búsqueda mucho más rápida que aprovecha la ordenación

*Comparar con el valor que esté en el medio de la lista:  
Si es el que se busca, terminar  
Si no, si es mayor, buscar en la primera mitad de la lista  
Si no, si es menor, buscar en la segunda mitad de la lista  
Repetir hasta encontrarlo o no quede sublista donde buscar*

Buscamos el 12

↓ Elemento mitad



Luis Hernández Yáñez

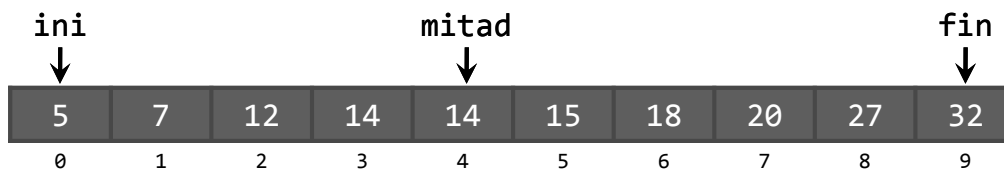


# Búsqueda binaria

Vamos buscando en sublistas cada vez más pequeñas (mitades)

Delimitamos el segmento de la lista donde buscar

Inicialmente tenemos toda la lista:



Índice del elemento en la mitad:  $\text{mitad} = (\text{ini} + \text{fin}) / 2$

Si no se encuentra, ¿dónde seguir buscando?

Buscado < elemento en la mitad:  $\text{fin} = \text{mitad} - 1$

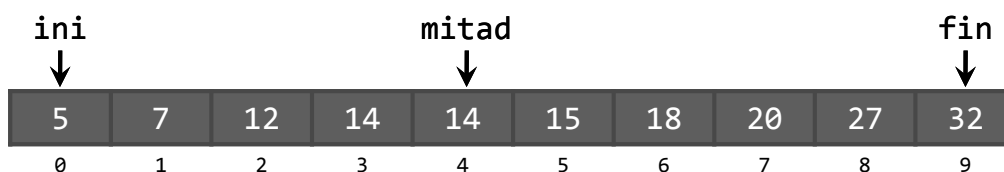
Buscado > elemento en la mitad:  $\text{ini} = \text{mitad} + 1$

Si  $\text{ini} > \text{fin}$ , no queda dónde buscar



# Búsqueda binaria

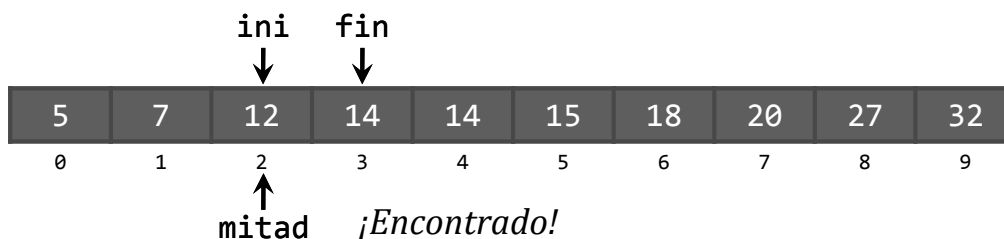
Buscamos el 12



$12 < \text{lista}[\text{mitad}] \rightarrow \text{fin} = \text{mitad} - 1$



$12 > \text{lista}[\text{mitad}] \rightarrow \text{ini} = \text{mitad} + 1$

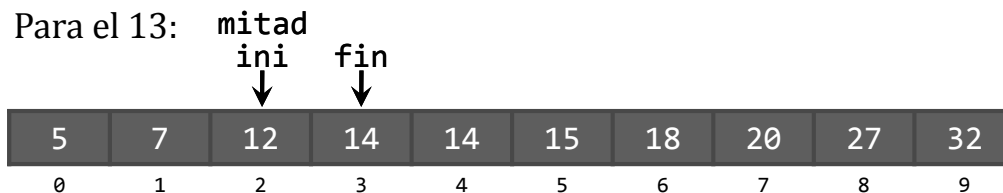




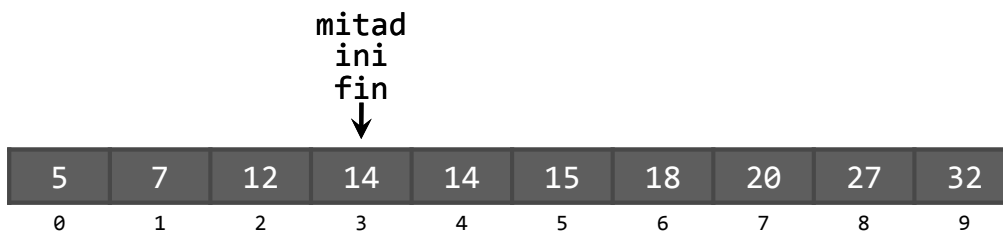
# Búsqueda binaria

Si el elemento no está, nos quedamos sin sublista:  $ini > fin$

Para el 13:



$13 > lista[mitad] \rightarrow ini = mitad + 1$



$13 < lista[mitad] \rightarrow fin = mitad - 1 \rightarrow 2$

!!!  $ini > fin$  !!! No hay dónde seguir buscando  $\rightarrow$  No está



# Búsqueda binaria

## Implementación

```
int buscado;
cout << "Valor a buscar: ";
cin >> buscado;
int ini = 0, fin = N - 1, mitad;
bool encontrado = false;
while ((ini <= fin) && !encontrado) {
    mitad = (ini + fin) / 2; // División entera
    if (buscado == lista[mitad]) {
        encontrado = true;
    }
    else if (buscado < lista[mitad]) {
        fin = mitad - 1;
    }
    else {
        ini = mitad + 1;
    }
} // Si se ha encontrado, está en [mitad]
```

```
const int N = 10;
typedef int tLista[N];
tLista lista;
```



# Búsqueda binaria

binaria.cpp

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

const int N = 100;
typedef int TArray[N];
typedef struct {
    TArray elementos;
    int cont;
} tLista;

int buscar(tLista lista, int buscado);

int main() {
    tLista lista;
    ifstream archivo;
    int dato;
    lista.cont = 0;
    archivo.open("ordenados.txt"); // Existe y es correcto
    archivo >> dato;
    ...
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Algoritmos de ordenación

Página 737



# Búsqueda binaria

```
while ((lista.cont < N) && (dato != -1)) {
    lista.elementos[lista.cont] = dato;
    lista.cont++;
    archivo >> dato;
}
archivo.close();
for (int i = 0; i < lista.cont; i++) {
    cout << lista.elementos[i] << " ";
}
cout << endl;
int buscado, pos;
cout << "Valor a buscar: ";
cin >> buscado;
pos = buscar(lista, buscado);
if (pos != -1) {
    cout << "Encontrado en la posición " << pos + 1 << endl;
}
else {
    cout << "No encontrado!" << endl;
}
return 0;
} ...
```

Luis Hernández Yáñez



Fundamentos de la programación: Algoritmos de ordenación

Página 738



# Búsqueda binaria

```
int buscar(tLista lista, int buscado) {
    int pos = -1, ini = 0, fin = lista.cont - 1, mitad;
    bool encontrado = false;
    while ((ini <= fin) && !encontrado) {
        mitad = (ini + fin) / 2; // División entera
        if (buscado == lista.elementos[mitad]) {
            encontrado = true;
        }
        else if (buscado < lista.elementos[mitad]) {
            fin = mitad - 1;
        }
        else {
            ini = mitad + 1;
        }
    }
    if (encontrado) {
        pos = mitad;
    }
    return pos;
}
```

Luis Hernández Yáñez



# Búsqueda binaria

## Complejidad

*¿Qué orden de complejidad tiene la búsqueda binaria?*

Caso peor:

No está o se encuentra en una sublista de 1 elemento

Nº de comparaciones = Nº de mitades que podemos hacer

$N / 2, N / 4, N / 8, N / 16, \dots, 8, 4, 2, 1$

$\equiv 1, 2, 4, 8, \dots, N / 16, N / 8, N / 4, N / 2$

Si hacemos que N sea igual a  $2^k$ :

$2^0, 2^1, 2^2, 2^3, \dots, 2^{k-4}, 2^{k-3}, 2^{k-2}, 2^{k-1}$

Nº de elementos de esa serie:  $k$

Nº de comparaciones =  $k$      $N = 2^k \rightarrow k = \log_2 N$

Complejidad:  $O(\log_2 N)$     Mucho más rápida que  $O(N)$

Luis Hernández Yáñez








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.





ANEXO

## Más sobre ordenación

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



---

## Índice

Ordenación por intercambio	744
Mezcla de dos listas ordenadas	747



## Ordenación por intercambio

Luis Hernández Yáñez

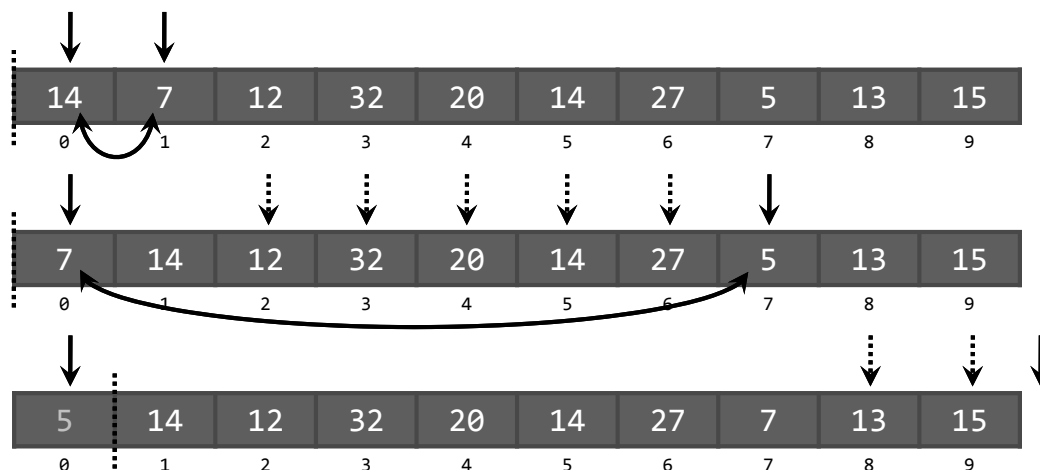


## Ordenación por intercambio

### *Algoritmo de ordenación por intercambio*

Variación del método de selección directa

Se intercambia el elemento de la posición que se trata en cada momento siempre que se encuentra uno que es menor:



Luis Hernández Yáñez



```
const int N = 10;
typedef int tLista[N];
tLista lista;
...
for (int i = 0; i < N - 1; i++) {
// Desde el primer elemento hasta el penúltimo
  for (int j = i + 1; j < N; j++) {
// Desde i+1 hasta el final
    if (lista[j] < lista[i]) {
      int tmp;
      tmp = lista[i];
      lista[i] = lista[j];
      lista[j] = tmp;
    }
  }
}
```

Igual número de comparaciones, muchos más intercambios  
No es estable



## Fundamentos de la programación

# Mezcla de dos listas ordenadas



# Mezcla de listas ordenadas

---

## *Mezcla de dos listas ordenadas en arrays*

```
const int N = 100;
typedef struct {
    int elementos[N];
    int cont;
} tLista;
```

Un índice para cada lista, inicializados a 0 (principio de las listas)

*Mientras que no lleguemos al final de alguna de las dos listas:*

*Elegimos el elemento menor de los que tienen los índices*

*Lo copiamos en la lista resultado y avanzamos su índice una posición*

*Copiamos en la lista resultado los que queden en la lista no acabada*



# Mezcla de listas ordenadas

---

```
void mezcla(tLista lista1, tLista lista2, tLista &listaM) {
    int pos1 = 0, pos2 = 0;
    listaM.cont = 0;

    while ((pos1 < lista1.cont) && (pos2 < lista2.cont)
           && (listaM.cont < N)) {
        if (lista1.elementos[pos1] < lista2.elementos[pos2]) {
            listaM.elementos[listaM.cont] = lista1.elementos[pos1];
            pos1++;
        }
        else {
            listaM.elementos[listaM.cont] = lista2.elementos[pos2];
            pos2++;
        }
        listaM.cont++;
    }
    ...
}
```





```
// Pueden quedar datos en alguna de las listas
if (pos1 < lista1.cont) {
    while ((pos1 < lista1.cont) && (listaM.cont < N)) {
        listaM.elementos[listaM.cont] = lista1.elementos[pos1];
        pos1++;
        listaM.cont++;
    }
}
else { // pos2 < lista2.cont
    while ((pos2 < lista2.cont) && (listaM.cont < N)) {
        listaM.elementos[listaM.cont] = lista2.elementos[pos2];
        pos2++;
        listaM.cont++;
    }
}
}
```

```
Primera lista:
1  4  5  8  12 12 15 18 24 31 45 49 63
Segunda lista:
2  3  9 14 15 23 28 42 58 73 79 84 88 93
Lista con la mezcla:
1  2  3  4  5  8  9 12 12 14 15 15 18 23 24 28 31 42
45 49 58 63 73 79 84 88 93
```



# Mezcla de listas ordenadas

## *Mezcla de dos listas ordenadas en archivos*

```
void mezcla(string nombre1, string nombre2, string nombreM) {
    // Mezcla las secuencias en los archivos nombre1 y nombre2
    // generando la secuencia mezclada en el archivo nombreM
    ifstream archivo1, archivo2;
    ofstream mezcla;
    int dato1, dato2;

    // Los archivos existen y son correctos
    archivo1.open(nombre1.c_str());
    archivo2.open(nombre2.c_str());
    mezcla.open(nombreM.c_str());
    archivo1 >> dato1;
    archivo2 >> dato2;
    while ((dato1 != -1) && (dato2 != -1)) {
        // Mientras quede algo en ambos archivos
        ...
    }
}
```



# Mezcla de listas ordenadas

---

```
if (dato1 < dato2) {
    mezcla << dato1 << endl;
    archivo1 >> dato1;
} else {
    mezcla << dato2 << endl;
    archivo2 >> dato2;
}
} // Uno de los dos archivos se ha acabado
if (dato1 != -1) { // Quedan en el primer archivo
    while (dato1 != -1) {
        mezcla << dato1 << endl;
        archivo1 >> dato1;
    }
}
else { // Quedan en el segundo archivo
    while (dato2 != -1) {
        mezcla << dato2 << endl;
        archivo2 >> dato2;
    }
}
...
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Algoritmos de ordenación (Anexo)

Página 752



# Mezcla de listas ordenadas

---

mezcla2.cpp

```
archivo2.close();
archivo1.close();
mezcla << -1 << endl;
mezcla.close();
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Algoritmos de ordenación (Anexo)

Página 753








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.





## Programación modular

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



---

## Índice

---

Programas multiarchivo y compilación separada	757
Interfaz frente a implementación	762
Uso de módulos de biblioteca	768
Ejemplo: Gestión de una lista ordenada I	770
Compilación de programas multiarchivo	778
El preprocesador	780
Cada cosa en su módulo	782
Ejemplo: Gestión de una lista ordenada II	784
El problema de las inclusiones múltiples	789
Compilación condicional	794
Protección frente a inclusiones múltiples	795
Ejemplo: Gestión de una lista ordenada III	796
Implementaciones alternativas	804
Espacios de nombres	808
Implementaciones alternativas	817
Calidad y reutilización del software	827



## Programas multiarchivo y compilación separada

Luis Hernández Yáñez



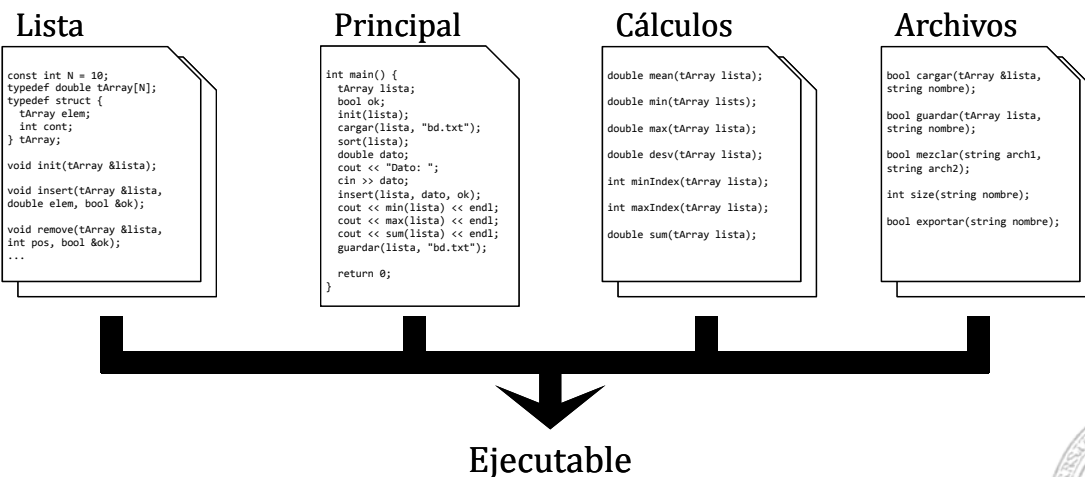
## Programación modular

### *Programas multiarchivo*

Código fuente repartido entre varios archivos (*módulos*)

Cada módulo con sus declaraciones y sus subprogramas

→ Módulo: Unidad funcional (estructura de datos, utilidades, ...)



Luis Hernández Yáñez



# Programación modular

## Compilación separada

Cada módulo se compila a código objeto de forma independiente

### Lista

```
const int N = 10;
typedef double TArray[N];
typedef struct {
    TArray elem;
    int cont;
} TArray;

void init(TArray &lista);

void insert(TArray &lista,
double elem, bool &ok);

void remove(TArray &lista,
int pos, bool &ok);
...
```

### lista.obj

```
00101110101011001010010010101
0010101001010101111010101000
101001010101010010101010101
011001010101010101010101001
01010101010000010101010101
01001010101010101000010101011
110010101010111100110010101
011010101010010010100111
00101010100101001010010
10100101010010100001001110
1001010110010101010010100
101010101010010101001010101
0100010101100101010010100
0110101110101001010100101
0101111010101001010101011
0000101010010101010101010
```

### Archivos

```
bool cargar(TArray &lista,
string nombre);

bool guardar(TArray lista,
string nombre);

bool mezclar(string arch1,
string arch2);

int size(string nombre);

bool exportar(string nombre);
```

### archivos.obj

```
11101010110010100101010010
1010010101011110101000010
01010101010010101010101010
01010101010101010100101010
01010100000101010101010100
10101010100001010101011100
1010101011100100101010110
101010100100101010011100010
10101000100100101010101010
01010001000010011101001
010110010101001010101001010
10101010010100010101010100
0010101110010101001000111
0101011001010100101010101
1111101011001101010110000
1001010010101010101010101
```

### Cálculos

```
double mean(TArray lista);
double min(TArray lista);
double max(TArray lista);
double desv(TArray lista);
int minIndex(TArray lista);
int maxIndex(TArray lista);
double sum(TArray lista);
```

### calculos.obj

```
01011001010010010101001010100
1010101111010101000101001010
1010101001010101010101001010
1010101010101000101010101010
101000010101010101010010101
010101000001010101110010101
010101110011001010101010101
01010010010100111001010101
010010100101010010101001010
1010010100001001110100101010
11001010100101010100101010101
0101001010101010101010000101
010110010101001010001101010
1110100101010010101011111
10101011001101010111000010010
101001010101010101000111010
```

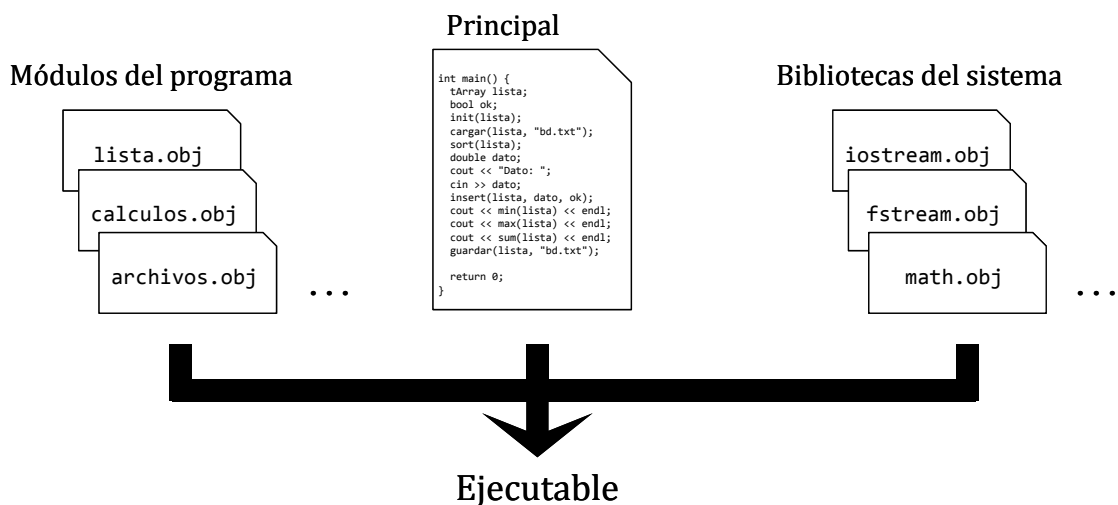
Luis Hernández Yáñez



# Programación modular

## Compilación separada

Al compilar el programa principal, se adjuntan los módulos compilados



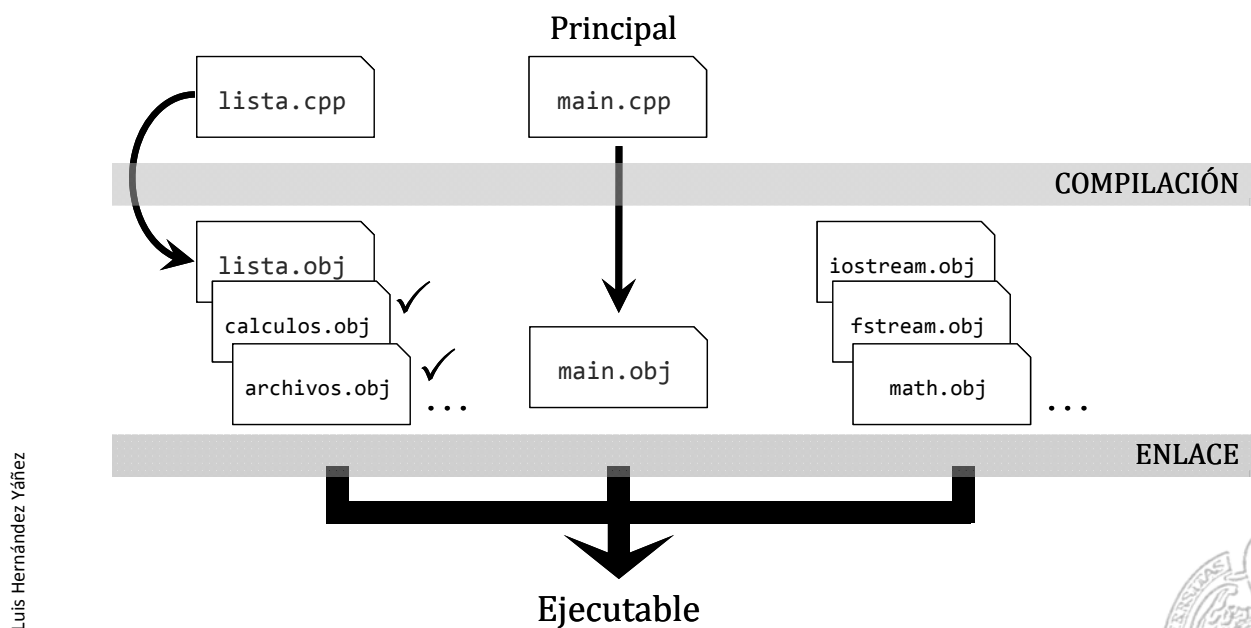
Luis Hernández Yáñez



# Programación modular

## Compilación separada

*¡Sólo los archivos fuente modificados necesitan ser recompilados!*



Luis Hernández Yáñez



# Fundamentos de la programación

## Interfaz frente a implementación

Luis Hernández Yáñez



# Interfaz frente a implementación

---

## Creación de módulos de biblioteca

Código de un programa de un único archivo:

- ✓ Definiciones de constantes
- ✓ Declaraciones de tipos de datos
- ✓ Prototipos de los subprogramas
- ✓ Implementación de los subprogramas
- ✓ Implementación de la función `main()`

Constantes, tipos y prototipos indican *cómo se usa*: Interfaz

- ✓ Estructura de datos con los subprogramas que la gestionan
  - ✓ Conjunto de utilidades (subprogramas) de uso general
  - ✓ Etcétera
- + Implementación de los subprogramas (*cómo se hace*)



# Interfaz frente a implementación

---

## Creación de módulos de biblioteca

Interfaz: Definiciones/declaraciones de datos y prototipos

¡Todo lo que el usuario de la unidad funcional necesita saber!

Implementación: Código de los subprogramas que hacen el trabajo

No hay que conocerlo para usarlo: ¡Seguro que es correcto!

Interfaz e implementación en dos archivos separados:

- ✓ Cabecera: Definiciones/declaraciones de datos y prototipos
- ✓ Implementación: Implementación de los subprogramas.

Archivo de cabecera: extensión `.h`

Archivo de implementación: extensión `.cpp`

} Mismo nombre

Repartimos el código entre ambos archivos (`lista.h/lista.cpp`)





# Interfaz frente a implementación

## Creación de módulos de biblioteca

### Interfaz frente a implementación

<pre>lista.h const int N = 10; typedef double tArray[N]; typedef struct {     tArray elem;     int cont; } tArray;  void init(tArray &amp;lista); void insert(tArray &amp;lista, double elem, bool &amp;ok); void remove(tArray &amp;lista, int pos, bool &amp;ok); ...</pre>	<pre>lista.cpp #include "lista.h" void init(tArray &amp;lista) {     lista.cont = 0; } void insert(tArray &amp;lista, double elem, bool &amp;ok) {     if (lista.cont == N) {         ok false;     }     else {         ...     } }</pre>	Módulo Unidad Biblioteca
---	--	--------------------------------

Si otro módulo quiere usar algo de esa biblioteca:  
Debe incluir el archivo de cabecera

main.cpp

```
#include "lista.h"
...
```

Los nombres de archivos de cabecera propios (no del sistema) se encierran entre dobles comillas, no entre ángulos

Luis Hernández Yáñez



# Interfaz frente a implementación

## Creación de módulos de biblioteca

### Interfaz

Archivo de cabecera (.h): todo lo que necesita conocer otro módulo (o programa principal) que quiera utilizar sus servicios (subprogramas)

La directiva `#include` añade las declaraciones del archivo de cabecera en el código del módulo (*preprocesamiento*):

main.cpp

```
#include "lista.h"
...
```

Preprocesador



main.cpp

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elem;
    int cont;
} tArray;

void init(tArray &lista);

void insert(tArray &lista, double elem,
bool &ok);

void remove(tArray &lista, int pos,
bool &ok);
...
```

Todo lo que se necesita saber para comprobar si el código de `main.cpp` hace un uso correcto de la lista (declaraciones y llamadas)

lista.h

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elem;
    int cont;
} tArray;

void init(tArray &lista);
void insert(tArray &lista,
double elem, bool &ok);
void remove(tArray &lista,
int pos, bool &ok);
...
```

Luis Hernández Yáñez



# Interfaz frente a implementación

## Creación de módulos de biblioteca

### Implementación

Compilar el módulo significa compilar su archivo de implementación (.cpp)

También necesita conocer sus propias declaraciones:

lista.cpp

```
#include "lista.h"  
...
```

lista.cpp

```
#include "lista.h"  
void init(tArray &lista) {  
    lista.cont = 0;  
}  
  
void insert(tArray &lista,  
double elem, bool &ok) {  
    if (lista.cont == N) {  
        ok = false;  
    }  
    else {  
        ...  
    }  
}
```

lista.obj

```
00101110101011001010010010101  
0010101001010101111010101000  
1010010101010100010101010101  
0110010101010101010101010001  
01010101010000010101010101  
010010101010101000010101011  
1100101010101110010010101  
011010101010010010100111  
0010101010010100101001010010  
101001010100101000010011110  
10010101010010101010010100  
101010101010010101001010101  
010001010101100101010010100  
011010101101001101010100101  
01011111010101001101010111  
00001001010100101010101010
```

Al compilar el módulo se genera el código objeto

Si no se modifica no hay necesidad de recompilar

Código que usa el módulo:

- ✓ Necesita sólo el archivo de cabecera para compilar
- ✓ Se adjunta el código objeto del módulo durante el enlace



# Fundamentos de la programación

## Uso de módulos de biblioteca



# Programación modular

## Uso de módulos de biblioteca

Ejemplo: Gestión de una lista ordenada (Tema 7)

Todo lo que tenga que ver con la lista estará en su propio módulo

Ahora el código estará repartido en tres archivos:

- ✓ lista.h: archivo de cabecera del módulo de lista
- ✓ lista.cpp: implementación del módulo de lista
- ✓ bd.cpp: programa principal que usa la lista

Tanto lista.cpp como bd.cpp deben incluir al principio lista.h

Módulo propio: dobles comillas en la directiva #include

#include "lista.h"

Archivos de cabecera de bibliotecas del sistema: entre ángulos

Y no tienen necesariamente que llevar extensión .h



# Programación modular

Archivo de cabecera

lista.h

## Módulo: Gestión de una lista ordenada I

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

const int N = 100;
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
const string BD = "bd.txt";
...
```

```
1 #include <string>
2 using namespace std;
3
4 const int N = 100;
5
6 // Estructura para los datos individuales de la lista:
7 typedef struct {
8     int codigo;
9     string nombre;
10    double sueldo;
11 } tRegistro;
12
13 // Array de registros:
14 typedef tRegistro tArray[N];
15
16 // Lista: array y contador
17 typedef struct {
18     tArray registros;
19     int cont;
20 } tLista;
21
22 // Constante global con el nombre del archivo de base de datos:
23 const string BD = "bd.txt";
24
25 // Muestra en una línea la información del registro proporcionado
26 // precedida por su posición en la lista.
27 void mostrar(int pos, tRegistro registro);
28
29 // Muestra la lista completa.
30 void mostrar(const tLista &lista);
31
32 // Operador relacional para comparar registros.
33 // Basado en el campo nombre.
34 bool operator<(tRegistro opIzq, tRegistro opDer);
35
36 // Operador relacional para comparar registros.
37 // Basado en el campo nombre.
38 bool operator<=(tRegistro opIzq, tRegistro opDer);
39
40 // Lectura de los datos de un nuevo registro.
41 tRegistro nuevo();
```

¡Documenta bien el código!



# Programación modular

---

```
void mostrar(int pos, tRegistro registro);
void mostrar(const tLista &lista);
bool operator>(tRegistro opIzq, tRegistro opDer);
bool operator<(tRegistro opIzq, tRegistro opDer);
tRegistro nuevo();
void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N
int buscar(tLista lista, string nombre);
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
```

Cada prototipo, con un comentario que explique su utilidad/uso  
(Aquí se omiten por cuestión de espacio)



# Programación modular

Implementación

lista.cpp

## *Módulo: Gestión de una lista ordenada I*

```
#include <iostream>
#include <string>
using namespace std;
#include <fstream>
#include <iomanip>
#include "lista.h"

tRegistro nuevo() {
    tRegistro registro;
    cout << "Introduce el código: ";
    cin >> registro.codigo;
    cout << "Introduce el nombre: ";
    cin >> registro.nombre;
    cout << "Introduce el sueldo: ";
    cin >> registro.sueldo;
    return registro;
} ...
```



# Programación modular

---

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        }
        // Insertamos en la posición i
        for (int j = lista.cont; j > i; j--) {
            // Desplazamos a la derecha
            lista.registros[j] = lista.registros[j - 1];
        }
        lista.registros[i] = registro;
        lista.cont++;
    }
} ...
```

Luis Hernández Yáñez



# Programación modular

---

```
void eliminar(tLista &lista, int pos, bool &ok) { // pos = 1..
    ok = true;
    if ((pos < 1) || (pos > lista.cont)) {
        ok = false; // Posición inexistente
    }
    else {
        pos--; // Pasamos a índice del array
        for (int i = pos + 1; i < lista.cont; i++) {
            // Desplazamos a la izquierda
            lista.registros[i - 1] = lista.registros[i];
        }
        lista.cont--;
    }
} ...
```

Luis Hernández Yáñez



## *Módulo: Gestión de una lista ordenada I*

```
#include <iostream>
using namespace std;
#include "lista.h"

int menu();

int main() {
    tLista lista;
    bool ok;
    int op, pos;
    cargar(lista, ok);
    if (!ok) {
        cout << "No se ha podido abrir el archivo!" << endl;
    }
    else {
        do {
            mostrar(lista);
            op = menu(); ...
        }
    }
}
```



# Programación modular

```
if (op == 1) {
    tRegistro registro = nuevo();
    insertar(lista, registro, ok);
    if (!ok) {
        cout << "Error: Lista llena!" << endl;
    }
}
else if (op == 2) {
    cout << "Posición: ";
    cin >> pos;
    eliminar(lista, pos, ok);
    if (!ok) {
        cout << "Error: Posicion inexistente!" << endl;
    }
}
else if (op == 3) {
    string nombre;
    cin.sync();
    cout << "Nombre: ";
    cin >> nombre;
    int pos = buscar(lista, nombre);
    ...
}
```



# Programación modular

---

```
        if (pos == -1) {
            cout << "No se ha encontrado!" << endl;
        }
        else {
            cout << "Encontrado en la posición " << pos << endl;
        }
    }
} while (op != 0);
guardar(lista);
}
return 0;
}

int menu() {
    cout << endl;
    cout << "1 - Insertar" << endl;
    cout << "2 - Eliminar" << endl;
    cout << "3 - Buscar" << endl;
    cout << "0 - Salir" << endl;
    int op;
    do {
        ...
    }
}
```



# Fundamentos de la programación

---

## Compilación de programas multiarchivo



# Compilación de programas multiarchivo

---

## G++

Archivos de cabecera e implementación en la misma carpeta

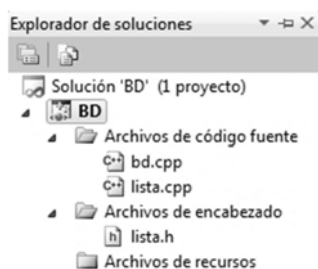
Listamos todos los .cpp en la orden g++:

```
D:\FP\Tema08>g++ -o bd.exe lista.cpp bd.cpp
```

Recuerda que sólo se compilan los .cpp

## Visual C++/Studio

Archivos de cabecera e implementación en grupos distintos:



A los archivos de cabecera los llama de encabezado

Con Depurar -> Generar solución se compilan todos los .cpp



---

# Fundamentos de la programación

## El preprocesador





# El preprocesador

---

Directivas: # . . .

Antes de compilar se pone en marcha el *preprocesador*

Interpreta las directivas y genera un único archivo temporal con todo el código del módulo o programa

Como en la inclusión (directiva `#include`):

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

const int N = 100;

typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

typedef tRegistro
tArray[N];

typedef struct {
    tArray registros;
    int cont;
} tLista;
...
```

```
#include "lista.h"

int menu();
...
```

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

const int N = 100;

typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

typedef tRegistro
tArray[N];

typedef struct {
    tArray registros;
    int cont;
} tLista;
...

int menu();
...
```

Luis Hernández Yáñez



# Fundamentos de la programación

---

## Cada cosa en su módulo

Luis Hernández Yáñez



# Programación modular

## *Distribuir la funcionalidad del programa en módulos*

Encapsulación de un conjunto de subprogramas relacionados:

- ✓ Por la estructura de datos sobre la que trabajan
- ✓ Subprogramas de utilidad

A menudo las estructuras de datos contienen otras estructuras:

```
const int N = 100;
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
```

Lista de registros:

- ✓ Estructura tRegistro
- ✓ Estructura tLista  
(contiene tRegistro)

Cada estructura, en su módulo



## Módulo de registros

Cabecera **registro.h**

### *Gestión de una lista ordenada II*

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

typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

tRegistro nuevo();
bool operator>(tRegistro opIzq, tRegistro opDer);
bool operator<(tRegistro opIzq, tRegistro opDer);
void mostrar(int pos, tRegistro registro);
```



## Gestión de una lista ordenada II

```
#include <iostream>
#include <string>
using namespace std;
#include <iomanip>
#include "registro.h" ←

tRegistro nuevo() {
    tRegistro registro;
    cout << "Introduce el código: ";
    cin >> registro.codigo;
    cout << "Introduce el nombre: ";
    cin >> registro.nombre;
    cout << "Introduce el sueldo: ";
    cin >> registro.sueldo;
    return registro;
}

bool operator>(tRegistro opIzq, tRegistro opDer) {
    return opIzq.nombre > opDer.nombre;
} ...
```

Luis Hernández Yáñez



## Gestión de una lista ordenada II

```
#include <string>
using namespace std;
#include "registro.h" ←

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
const string BD = "bd.txt";

void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N
int buscar(tLista lista, string nombre);
void mostrar(const tLista &lista);
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
```

Luis Hernández Yáñez



## Gestión de una lista ordenada II

```
#include <iostream>
using namespace std;
#include <fstream>
#include "lista2.h" ←

void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        }
        // Insertamos en la posición i
        for (int j = lista.cont; j > i; j--) { // Desplazar a la derecha
            lista.registros[j] = lista.registros[j - 1];
        }
        ...
    }
}
```

Luis Hernández Yáñez



# Programa principal

## Gestión de una lista ordenada II

```
#include <iostream>
using namespace std;
#include "registro.h" ←
#include "lista2.h" ←

int menu();

int main() {
    tLista lista;
    bool ok;
    int op, pos;

    cargar(lista, ok);
    if (!ok) {
        cout << "No se pudo abrir el archivo!" << endl;
    }
    else {
        do {
            mostrar(lista);
            op = menu();
            ...
        }
    }
}
```



*¡No intentes compilar este ejemplo!*

*Tiene errores*

Luis Hernández Yáñez



## El problema de las inclusiones múltiples

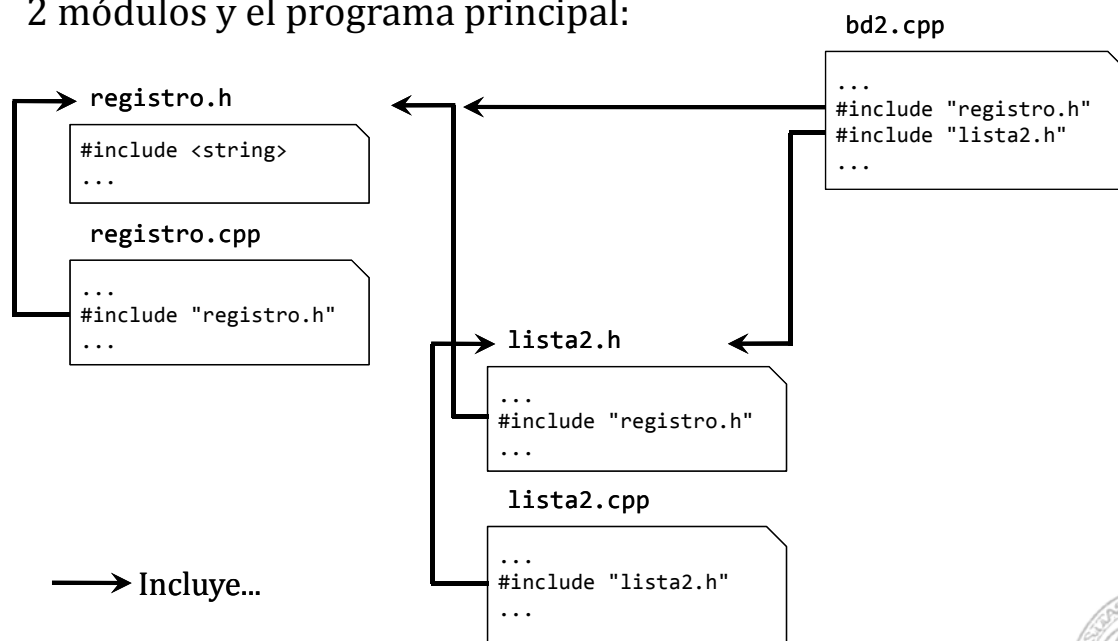
Luis Hernández Yáñez



## Inclusiones múltiples

### *Gestión de una lista ordenada II*

2 módulos y el programa principal:



Luis Hernández Yáñez



# Inclusiones múltiples

## Gestión de una lista ordenada II

Preprocesamiento de #include:

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

```
#include "registro.h"
```

```
#include "lista2.h"
```

```
int menu();
```

```
...
```

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

```
typedef struct {
    ...
} tRegistro;
...
```

```
#include <string>
using namespace std;
#include "registro.h"

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
...
```

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

```
typedef struct {
    ...
} tRegistro;
...
```

Luis Hernández Yáñez



# Inclusiones múltiples

## Gestión de una lista ordenada II

Preprocesamiento de #include:

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

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

typedef struct {
    ...
} tRegistro;
...
```

```
#include "lista2.h"
```

```
int menu();
```

```
...
```



Sustituido

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

typedef struct {
    ...
} tRegistro;
...

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
...
```

Luis Hernández Yáñez



# Inclusiones múltiples

## Gestión de una lista ordenada II

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

#include <string>
using namespace std;

typedef struct {
    ...
} tRegistro;

...

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

typedef struct {
    ...
} tRegistro;

...

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;

...

int menu();

...
```

👁️ ¡Identificador duplicado!

Luis Hernández Yáñez



# Inclusiones múltiples

## Compilación condicional

Directivas `#ifdef`, `#ifndef`, `#else` y `#endif`

Se usan en conjunción con la directiva `#define`

<pre>#define X #ifdef X ... // Código if [#else ... // Código else ] #endif</pre>	<pre>#define X #ifndef X ... // Código if [#else ... // Código else ] #endif</pre>
---	--

La directiva `#define` define un símbolo (identificador)

Izquierda: se compilará el “Código if” y no el “Código else”

Derecha: al revés, o nada si no hay else

Las cláusulas else son opcionales

Luis Hernández Yáñez



# Inclusiones múltiples

## *Protección frente a inclusiones múltiples*

lista2.cpp y bd2.cpp incluyen registro.h

→ ¡Identificadores duplicados!

Cada módulo debe incluirse una y sólo una vez

Protección frente a inclusiones múltiples:

```
#ifndef X
#define X
... // Módulo
#endif
```



*El símbolo X debe ser único para cada módulo de la aplicación*

La primera vez no está definido el símbolo X: se incluye y define

Las siguientes veces el símbolo X ya está definido: no se incluye

Símbolo X: nombre del archivo con \_ en lugar de .

registro\_h, lista2\_h, ...



# Módulo de registros

Cabecera **registrofin.h**

## *Gestión de una lista ordenada III*

```
#ifndef registrofin_h
#define registrofin_h
#include <string>
using namespace std;

typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

tRegistro nuevo();
bool operator>(tRegistro opIzq, tRegistro opDer);
bool operator<(tRegistro opIzq, tRegistro opDer);
void mostrar(int pos, tRegistro registro);
#endif
```





## Gestión de una lista ordenada III

```
#include <iostream>
#include <string>
using namespace std;
#include <iomanip>
#include "registrofin.h" ←

tRegistro nuevo() {
    tRegistro registro;
    cout << "Introduce el código: ";
    cin >> registro.codigo;
    cout << "Introduce el nombre: ";
    cin >> registro.nombre;
    cout << "Introduce el sueldo: ";
    cin >> registro.sueldo;
    return registro;
}

bool operator>(tRegistro opIzq, tRegistro opDer) {
    return opIzq.nombre > opDer.nombre;
} ...
```

Luis Hernández Yáñez



## Gestión de una lista ordenada III

```
#ifndef listafin_h
#define listafin_h
#include <string>
using namespace std;
#include "registrofin.h" ←

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
const string BD = "bd.txt";
void mostrar(const tLista &lista);
void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N
int buscar(tLista lista, string nombre);
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
#endif
```

Luis Hernández Yáñez



## Gestión de una lista ordenada III

```
#include <iostream>
using namespace std;
#include <fstream>
#include "listafin.h" ←

void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        }
        // Insertamos en la posición i
        for (int j = lista.cont; j > i; j--) {
            // Desplazamos a la derecha
            lista.registros[j] = lista.registros[j - 1];
        }
        ...
    }
}
```

Luis Hernández Yáñez



# Programa principal

## Gestión de una lista ordenada III

```
#include <iostream>
using namespace std;
#include "registrofin.h" ←
#include "listafin.h" ←

int menu();

int main() {
    tLista lista;
    bool ok;
    int op, pos;

    cargar(lista, ok);
    if (!ok) {
        cout << "No se pudo abrir el archivo!" << endl;
    }
    else {
        do {
            mostrar(lista);
            op = menu();
            ...
        }
    }
}
```

**¡Ahora ya puedes compilarlo!**

Luis Hernández Yáñez



# Inclusiones múltiples

## Gestión de una lista ordenada III

Preprocesamiento de #include en bdfin.cpp:

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

#include "registrofin.h"

#include "listafin.h"

int menu();

...
```

```
#ifndef registrofin_h
#define registrofin_h
#include <string>
using namespace std;

typedef struct {
    ...
} tRegistro;
...
```

👁️ registrofin\_h no se ha definido todavía

Luis Hernández Yáñez



# Inclusiones múltiples

## Gestión de una lista ordenada III

Preprocesamiento de #include en bdfin.cpp:

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

#define registrofin_h
#include <string>
using namespace std;

typedef struct {
    ...
} tRegistro;
...

#include "listafin.h"

int menu();

...
```

```
#ifndef listafin_h
#define listafin_h
#include <string>
using namespace std;
#include "registrofin.h"

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
...
```

👁️ listafin\_h no se ha definido todavía

Luis Hernández Yáñez



# Inclusiones múltiples

## Gestión de una lista ordenada III

Preprocesamiento de #include en bdfin.cpp:


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

```
typedef struct {
    ...
} tRegistro;
...
```

```
#define listafin_h
#include <string>
using namespace std;
#include "registrofin.h"
...
int menu();
...
```

```
#ifndef registrofin_h
#define registrofin_h
#include <string>
using namespace std;

typedef struct {
    ...
} tRegistro;
...
```

 ;registrofin\_h ya está definido!



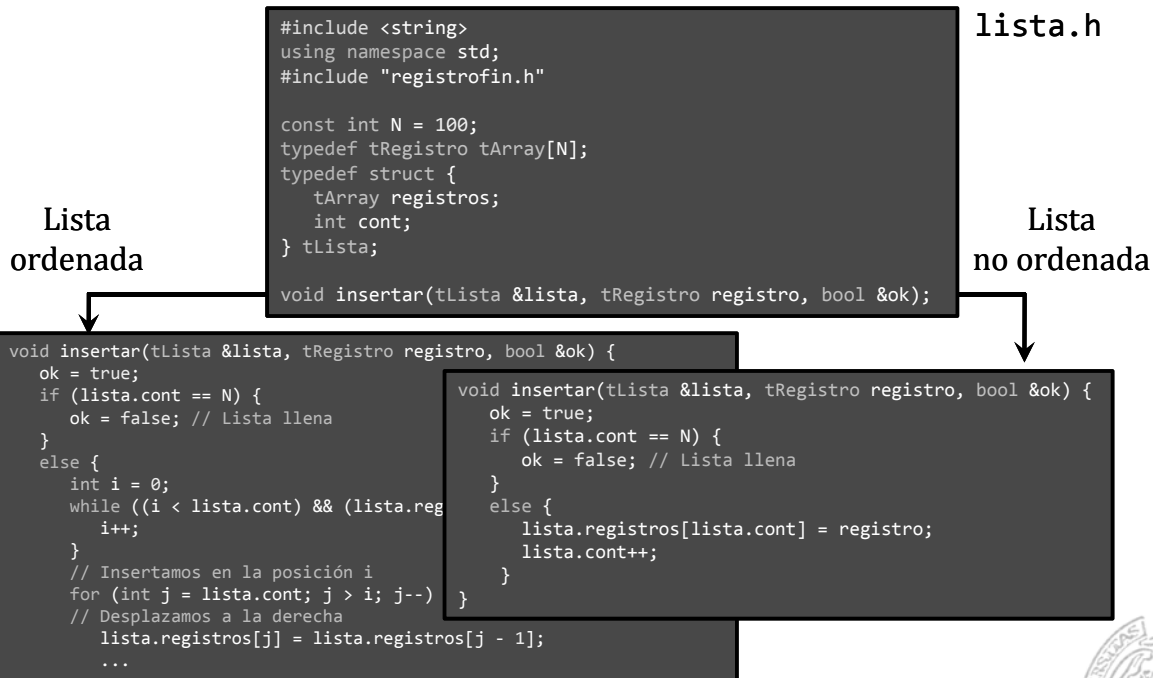
# Fundamentos de la programación

## Implementaciones alternativas



# Implementaciones alternativas

## Misma interfaz, implementación alternativa



Luis Hernández Yáñez



# Implementaciones alternativas

## Misma interfaz, implementación alternativa



Luis Hernández Yáñez



# Implementaciones alternativas

---

## *Misma interfaz, implementación alternativa*

Al compilar, incluimos un archivo de implementación u otro:  
¿Programa con lista ordenada o con lista desordenada?

```
g++ -o programa.exe registrofin.cpp listaORD.cpp ...
```

Incluye la implementación de la lista con ordenación

```
g++ -o programa.exe registrofin.cpp listaDES.cpp ...
```

Incluye la implementación de la lista sin ordenación



# Fundamentos de la programación

---

## Espacios de nombres



# Espacios de nombres

---

## *Agrupaciones lógicas de declaraciones*

*Espacio de nombres*: agrupación de declaraciones (tipos, datos, subprogramas) bajo un nombre distintivo

Forma de un espacio de nombres:

```
namespace nombre {  
    // Declaraciones  
}
```

Por ejemplo:

```
namespace miEspacio {  
    int i;  
    double d;  
}
```

Variables *i* y *d* declaradas en el espacio de nombres *miEspacio*



# Espacios de nombres

---

## *Acceso a miembros de un espacio de nombres*

*Operador de resolución de ámbito* (::)

Acceso a las variables del espacio de nombres *miEspacio*:

Nombre del espacio y operador de resolución de ámbito

```
miEspacio::i
```

```
miEspacio::d
```

Puede haber entidades con el mismo identificador en distintos módulos o en ámbitos distintos de un mismo módulo

Cada declaración en un espacio de nombres distinto:

```
namespace primero {  
    int x = 5;  
} namespace segundo {  
    double x = 3.1416;  
}
```

Ahora se distingue entre *primero::x* y *segundo::x*



# Espacios de nombres

## *using*

Introduce un nombre de un espacio de nombres en el ámbito actual:

```
#include <iostream>
using namespace std;
namespace primero {
    int x = 5;
    int y = 10;
}
namespace segundo {
    double x = 3.1416;
    double y = 2.7183;
}
int main() {
    using primero::x;
    using segundo::y;
    cout << x << endl; // x es primero::x
    cout << y << endl; // y es segundo::y
    cout << primero::y << endl; // espacio explícito
    cout << segundo::x << endl; // espacio explícito
    return 0;
}
```

```
5
2.7183
10
3.1416
```

Luis Hernández Yáñez



# Espacios de nombres

## *using namespace*

Introduce todos los nombres de un espacio en el ámbito actual:

```
#include <iostream>
using namespace std;
namespace primero {
    int x = 5;
    int y = 10;
}
namespace segundo {
    double x = 3.1416;
    double y = 2.7183;
}
int main() {
    using namespace primero;
    cout << x << endl; // x es primero::x
    cout << y << endl; // y es primero::y
    cout << segundo::x << endl; // espacio explícito
    cout << segundo::y << endl; // espacio explícito
    return 0;
}
```

```
using [namespace]
sólo tiene efecto
en el bloque
en que se encuentra
```

```
5
10
3.1416
2.7183
```

Luis Hernández Yáñez





# Ejemplo de espacio de nombres

---

```
#ifndef listaEN_h
#define listaEN_h
#include "registrofin.h"

namespace ord { // Lista ordenada
    const int N = 100;
    typedef tRegistro tArray[N];
    typedef struct {
        tArray registros;
        int cont;
    } tLista;
    const string BD = "bd.txt";
    void mostrar(const tLista &lista);
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    void eliminar(tLista &lista, int pos, bool &ok); // 1..N
    int buscar(tLista lista, string nombre);
    void cargar(tLista &lista, bool &ok);
    void guardar(tLista lista);
} // namespace

#endif
```



# Ejemplo de espacio de nombres

---

## *Implementación*

```
#include <iostream>
#include <fstream>
using namespace std;
#include "listaEN.h"

void ord::insertar(tLista &lista, tRegistro registro, bool &ok) {
    // ...
}

void ord::eliminar(tLista &lista, int pos, bool &ok) {
    // ...
}

int ord::buscar(tLista lista, string nombre) {
    // ...
}

...
```



# Ejemplo de espacio de nombres

---

## *Uso del espacio de nombres*

Quien utilice listaEN.h debe poner el nombre del espacio:

```
#include <iostream>
using namespace std;
#include "registrofin.h"
#include "listaEN.h"

int menu();

int main() {
    ord::tLista lista;
    bool ok;
    ord::cargar(lista, ok);
    if (!ok) {
        cout << "No se pudo abrir el archivo!" << endl;
    }
    else {
        ord::mostrar(lista);
        ...
    }
}
```

O usar una instrucción using namespace ord;

Luis Hernández Yáñez



# Ejemplo de espacio de nombres

---

## *Uso del espacio de nombres*

```
#include <iostream>
using namespace std;
#include "registrofin.h"
#include "listaEN.h"
using namespace ord; ←

int menu();

int main() {
    tLista lista;
    bool ok;
    cargar(lista, ok);
    if (!ok) {
        cout << "No se pudo abrir el archivo!" << endl;
    }
    else {
        mostrar(lista);
        ...
    }
}
```

Luis Hernández Yáñez



# Espacios de nombres

## *Implementaciones alternativas*

Distintos espacios de nombres para distintas implementaciones

¿Lista ordenada o lista desordenada?

```
namespace ord { // Lista ordenada
    const int N = 100;
    typedef tRegistro tArray[N];
    ...
    void mostrar(const tLista &lista);
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    ...
} // namespace

namespace des { // Lista desordenada
    const int N = 100;
    typedef tRegistro tArray[N];
    ...
    void mostrar(const tLista &lista);
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    ...
} // namespace
```

Luis Hernández Yáñez



## Ejemplo

Cabecera

listaEN.h

## *Implementaciones alternativas*

Todo lo común puede estar fuera de la estructura namespace:

```
#ifndef listaEN_H
#define listaEN_H

#include "registrofin.h"

const int N = 100;

typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;

void mostrar(const tLista &lista);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N

...
```

Luis Hernández Yáñez



# Implementaciones alternativas

```
namespace ord { // Lista ordenada
    const string BD = "bd.txt";
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    int buscar(tLista lista, string nombre);
    void cargar(tLista &lista, bool &ok);
    void guardar(tLista lista);
} // namespace
```

```
namespace des { // Lista desordenada
    const string BD = "bddes.txt";
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    int buscar(tLista lista, string nombre);
    void cargar(tLista &lista, bool &ok);
    void guardar(tLista lista);
} // namespace
```

#endif



cargar() y guardar() se distinguen porque usan su propia BD, pero se implementan exactamente igual



# Implementaciones alternativas

listaEN.cpp

```
#include <iostream>
using namespace std;
#include <fstream>
#include "listaEN.h"

// IMPLEMENTACIÓN DE LOS SUBPROGRAMAS COMUNES
void eliminar(tLista &lista, int pos, bool &ok) { // ...
}

void mostrar(const tLista &lista) { // ...
}

// IMPLEMENTACIÓN DE LOS SUBPROGRAMAS DEL ESPACIO DE NOMBRES ord
void ord::insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        } ...
    }
}
```



# Implementaciones alternativas

---

```
        for (int j = lista.cont; j > i; j--) {
            lista.registros[j] = lista.registros[j - 1];
        }
        lista.registros[i] = registro;
        lista.cont++;
    }
}

int ord::buscar(tLista lista, string nombre) {
    int ini = 0, fin = lista.cont - 1, mitad;
    bool encontrado = false;
    while ((ini <= fin) && !encontrado) {
        mitad = (ini + fin) / 2;
        if (nombre == lista.registros[mitad].nombre) {
            encontrado = true;
        }
        else if (nombre < lista.registros[mitad].nombre) {
            fin = mitad - 1;
        }
        else {
            ini = mitad + 1;
        }
    }
    ...
}
```

Luis Hernández Yáñez



# Implementaciones alternativas

---

```
    if (encontrado) {
        mitad++;
    }
    else {
        mitad = -1;
    }
    return mitad;
}

void ord::cargar(tLista &lista, bool &ok) { // ...
}

void ord::guardar(tLista lista) { // ...
}
...

```

Luis Hernández Yáñez



# Implementaciones alternativas

---

```
// IMPLEMENTACIÓN DE LOS SUBPROGRAMAS DEL ESPACIO DE NOMBRES des

void des::insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        lista.registros[lista.cont] = registro;
        lista.cont++;
    }
}

int des::buscar(tLista lista, string nombre) {
    int pos = 0;
    bool encontrado = false;
    while ((pos < lista.cont) && !encontrado) {
        if (nombre == lista.registros[pos].nombre) {
            encontrado = true;
        }
        else {
            pos++;
        }
    } ...
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 823



# Implementaciones alternativas

---

```
    if (encontrado) {
        pos++;
    }
    else {
        pos = -1;
    }
    return pos;
}

void des::cargar(tLista &lista, bool &ok) { // ...
}

void des::guardar(tLista lista) { // ...
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 824



## Programa principal

```
#include <iostream>
using namespace std;
#include "registrofin.h"
#include "listaEN.h"
using namespace ord;

int menu();

int main() {
    tLista lista;
    bool ok;
    ...
    tRegistro registro = nuevo();
    insertar(lista, registro, ok);
    if (!ok) {
        ...
    }
}
```

```
1: 12345 Alvarez 120000.00
2: 11111 Benitez 100000.00
3: 21112 Dominguez 90000.00
4: 11111 Duran 120000.00
5: 22222 Fernandez 120000.00
6: 12345 Gomez 100000.00
7: 10000 Hernandez 150000.00
8: 21112 Jimenez 100000.00
9: 54321 Manzano 95000.00
10: 11111 Perez 90000.00
11: 12345 Sanchez 90000.00
12: 10000 Sergei 100000.00
13: 33333 Tarazona 120000.00
14: 12345 Turegano 100000.00
15: 11111 Urpiano 90000.00

1 - Insertar
2 - Eliminar
3 - Buscar
0 - Salir
Opción: 1
Introduce el código: 33333
Introduce el nombre: Calvo
Introduce el sueldo: 95000
1: 12345 Alvarez 120000.00
2: 11111 Benitez 100000.00
3: 33333 Calvo 95000.00 ←
4: 21112 Dominguez 90000.00
5: 11111 Duran 120000.00
6: 22222 Fernandez 120000.00
7: 12345 Gomez 100000.00
8: 10000 Hernandez 150000.00
9: 21112 Jimenez 100000.00
10: 54321 Manzano 95000.00
11: 11111 Perez 90000.00
12: 12345 Sanchez 90000.00
13: 10000 Sergei 100000.00
14: 33333 Tarazona 120000.00
15: 12345 Turegano 100000.00
16: 11111 Urpiano 90000.00
```

Luis Hernández Yáñez



## Programa principal

```
#include <iostream>
using namespace std;
#include "registrofin.h"
#include "listaEN.h"
using namespace des;

int menu();

int main() {
    tLista lista;
    bool ok;
    ...
    tRegistro registro = nuevo();
    insertar(lista, registro, ok);
    if (!ok) {
        ...
    }
}
```

```
1: 12345 Alvarez 120000.00
2: 11111 Benitez 100000.00
3: 21112 Dominguez 90000.00
4: 11111 Duran 120000.00
5: 22222 Fernandez 120000.00
6: 12345 Gomez 100000.00
7: 10000 Hernandez 150000.00
8: 21112 Jimenez 100000.00
9: 54321 Manzano 95000.00
10: 11111 Perez 90000.00
11: 12345 Sanchez 90000.00
12: 10000 Sergei 100000.00
13: 33333 Tarazona 120000.00
14: 12345 Turegano 100000.00
15: 11111 Urpiano 90000.00

1 - Insertar
2 - Eliminar
3 - Buscar
0 - Salir
Opción: 1
Introduce el código: 33333
Introduce el nombre: Calvo
Introduce el sueldo: 95000
1: 12345 Alvarez 120000.00
2: 11111 Benitez 100000.00
3: 21112 Dominguez 90000.00
4: 11111 Duran 120000.00
5: 22222 Fernandez 120000.00
6: 12345 Gomez 100000.00
7: 10000 Hernandez 150000.00
8: 21112 Jimenez 100000.00
9: 54321 Manzano 95000.00
10: 11111 Perez 90000.00
11: 12345 Sanchez 90000.00
12: 10000 Sergei 100000.00
13: 33333 Tarazona 120000.00
14: 12345 Turegano 100000.00
15: 11111 Urpiano 90000.00
16: 33333 Calvo 95000.00
```

Luis Hernández Yáñez



## Calidad y reutilización del software



## Calidad del software

---

### *Software de calidad*

El software debe ser desarrollado con buenas prácticas de ingeniería del software que aseguren un buen nivel de calidad

Los distintos módulos de la aplicación deben ser probados exhaustivamente, tanto de forma independiente como en su relación con los demás módulos

La prueba y depuración es muy importante y todos los equipos deberán seguir buenas pautas para asegurar la calidad

Los módulos deben ser igualmente bien documentados, de forma que otros desarrolladores puedan aprovecharlos





# Prueba y depuración del software

---

## *Prueba exhaustiva*

El software debe ser probado exhaustivamente

Debemos intentar descubrir todos los errores posible

Los errores deben ser depurados, corrigiendo el código

Pruebas sobre listas:

- ✓ Lista inicialmente vacía
- ✓ Lista inicialmente llena
- ✓ Lista con un número intermedio de elementos
- ✓ Archivo no existente

Etcétera...

Se han de probar todas las opciones/situaciones del programa

En las clases prácticas veremos cómo se depura el software



# Reutilización del software

---

## *No reinventemos la rueda*

Desarrollar el software pensando en su posible reutilización

Un software de calidad debe poder ser fácilmente reutilizado

Nuestros módulos deben ser fácilmente usados y modificados

Por ejemplo: Nueva aplicación que gestione una lista de longitud variable de registros con NIF, nombre, apellidos y edad

Partiremos de los módulos `registro` y `lista` existentes

Las modificaciones básicamente afectarán al módulo `registro`








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.





ANEXO

## Ejemplo de modularización

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Modularización de un programa

ventas.cpp

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

const int NCLI = 100;
const int NPROD = 200;
const int NVENTAS = 3000;

typedef struct {
    int id_cli;
    string nif;
    string nombre;
    string telefono;
} tCliente;

typedef struct {
    tCliente clientes[NCLI];
    int cont;
} tListaClientes;

typedef struct {
    int id_prod;
    string codigo;
    string nombre;
    double precio;
    int unidades;
} tProducto;

typedef struct {
    tProducto productos[NPROD];
    int cont;
} tListaProductos;

typedef struct {
    int id;
    int id_prod;
    int id_cli;
    int unidades;
} tVenta;

typedef struct {
    tVenta ventas[NVENTAS];
    int cont;
} tListaVentas;

...
```



# Modularización de un programa

---

```
tCliente nuevoCliente();
bool valida(tCliente cliente); // Función interna
bool operator<(tCliente opIzq, tCliente opDer); // Por NIF
void mostrar(tCliente cliente);
void inicializar(tListaClientes &lista);
void cargar(tListaClientes &lista);
void insertar(tListaClientes &lista, tCliente cliente, bool &ok);
void buscar(const tListaClientes &lista, string nif, tCliente &cliente, bool &ok);
void eliminar(tListaClientes &lista, string nif, bool &ok);
void mostrar(const tListaClientes &lista);
tProducto nuevoProducto();
bool valida(tProducto producto); // Función interna
bool operator<(tProducto opIzq, tProducto opDer); // Por código
void mostrar(tProducto producto);
void inicializar(tListaProductos &lista);
void cargar(tListaProductos &lista);
void insertar(tListaProductos &lista, tProducto producto, bool &ok);
void buscar(const tListaProductos &lista, string codigo, tProducto &producto,
            bool &ok);
void eliminar(tListaProductos &lista, string codigo, bool &ok);
...
```



# Modularización de un programa

---

```
void mostrar(const tListaProductos &lista);
tVenta nuevaVenta(int id_prod, int id_cli, int unidades);
bool valida(tVenta venta); // Función interna
void mostrar(tVenta venta, const tListaClientes &clientes,
            const tListaProductos &productos);
void inicializar(tListaVentas &lista);
void cargar(tListaVentas &lista);
void insertar(tListaVentas &lista, tVenta venta, bool &ok);
void buscar(const tListaVentas &lista, int id, tVenta &venta, bool &ok);
void eliminar(tListaVentas &lista, int id, bool &ok);
void ventasPorClientes(const tListaVentas &lista);
void ventasPorProductos(const tListaVentas &lista);
double totalVentas(const tListaVentas &ventas, string nif,
                  const tListaClientes &clientes,
                  const tListaProductos &productos);
void stock(const tListaVentas &ventas, const tListaClientes &clientes,
          const tListaProductos &productos);
int menu();

int main() {
    ...
}
```



# Estructuras de datos

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

```
const int NCLI = 100;
const int NPROD = 200;
const int NVENTAS = 3000;
```

```
typedef struct {
    int id_cli;
    string nif;
    string nombre;
    string telefono;
} tCliente;
```

Cliente

Lista de clientes

```
typedef struct {
    tCliente clientes[NCLI];
    int cont;
} tListaClientes;
```

```
typedef struct {
    int id_prod;
    string codigo;
} tProducto;
```

Producto

```
string nombre;
double precio;
int unidades;
} tProducto;
```

Lista de productos

```
typedef struct {
    tProducto productos[NPROD];
    int cont;
} tListaProductos;
```

```
typedef struct {
    int id;
    int id_prod;
    int id_cli;
    int unidades;
} tVenta;
```

Venta

Lista de ventas

```
typedef struct {
    tVenta ventas[NVENTAS];
    int cont;
} tListaVentas;
```

...

Luis Hernández Yáñez



# Subprogramas de las estructuras de datos

```
tCliente nuevoCliente();
bool valida(tCliente cliente); // Función interna
bool operator<(tCliente opIzq, tCliente opDer); // Por NIF
void mostrar(tCliente cliente);
```

Cliente

Lista de clientes

```
void inicializar(tListaClientes &lista);
void cargar(tListaClientes &lista);
void insertar(tListaClientes &lista, tCliente cliente, bool &ok);
void buscar(const tListaClientes &lista, string nif, tCliente &cliente,
            bool &ok);
void eliminar(tListaClientes &lista, string nif, bool &ok);
void mostrar(const tListaClientes &lista);
```

```
tProducto nuevoProducto();
bool valida(tProducto producto); // Función interna
bool operator<(tProducto opIzq, tProducto opDer); // Por código
void mostrar(tProducto producto);
```

Producto

...

Luis Hernández Yáñez



# Subprogramas de las estructuras de datos

Lista de productos

```
void inicializar(tListaProductos &lista);
void cargar(tListaProductos &lista);
void insertar(tListaProductos &lista, tProducto producto, bool &ok);
void buscar(const tListaProductos &lista, string codigo, tProducto &producto,
            bool &ok);
void eliminar(tListaProductos &lista, string codigo, bool &ok);
void mostrar(const tListaProductos &lista);
```

```
tVenta nuevaVenta(int id_prod, int id_cli, int unidades);
bool valida(tVenta venta); // Función interna
void mostrar(tVenta venta, const tListaClientes &clientes,
            const tListaProductos &productos);
```

Venta

...



# Subprogramas de las estructuras de datos

Lista de ventas

```
void inicializar(tListaVentas &lista);
void cargar(tListaVentas &lista);
void insertar(tListaVentas &lista, tVenta venta, bool &ok);
void buscar(const tListaVentas &lista, int id, tVenta &venta, bool &ok);
void eliminar(tListaVentas &lista, int id, bool &ok);
void ventasPorClientes(const tListaVentas &lista);
void ventasPorProductos(const tListaVentas &lista);
double totalVentas(const tListaVentas &ventas, string nif,
                  const tListaClientes &clientes,
                  const tListaProductos &productos);
void stock(const tListaVentas &ventas, const tListaClientes &clientes,
           const tListaProductos &productos);
```

```
int menu();
```

```
int main() {
```

```
    ...
```



# Módulos

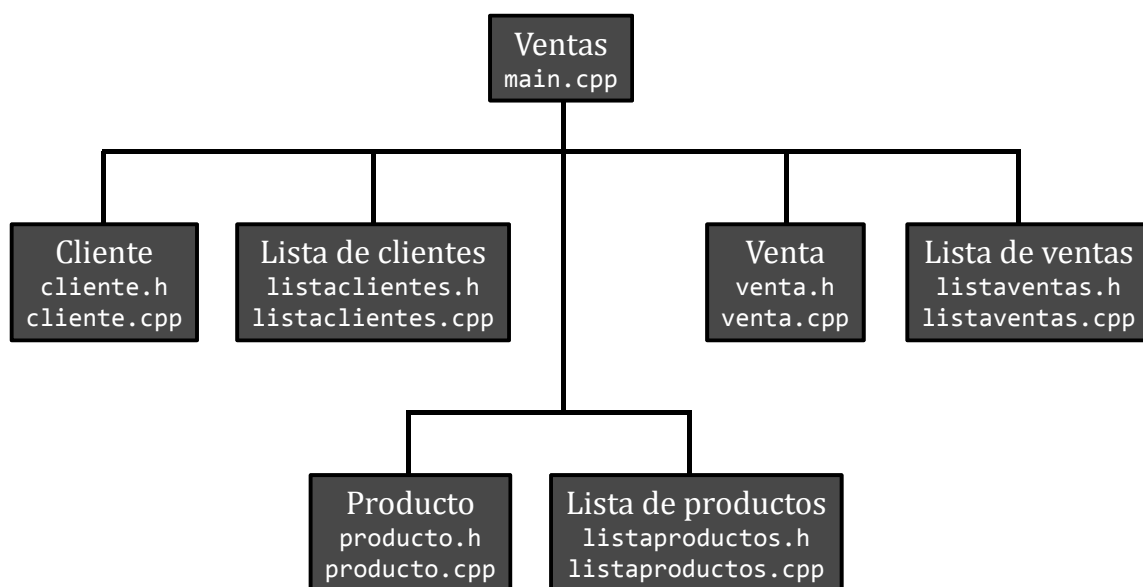
- ✓ Cliente: cliente.h y cliente.cpp
- ✓ Lista de clientes: listaclientes.h y listaclientes.cpp
- ✓ Producto: producto.h y producto.cpp
- ✓ Lista de productos: listaproductos.h y listaproductos.cpp
- ✓ Venta: venta.h y venta.cpp
- ✓ Lista de ventas: listaventas.h y listaventas.cpp
- ✓ Programa principal: main.cpp

Distribución del código en los módulos:

- ✓ Declaraciones de tipos y datos en el archivo de cabecera (.h)
- ✓ Prototipos en el archivo de cabecera (.h) (excepto los de los subprogramas privados –internos–, que irán en el .cpp)
- ✓ Implementación de los subprogramas en el .cpp



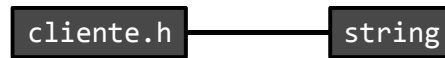
# Módulos



# Dependencias entre módulos

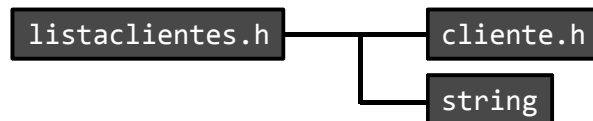
Inclusiones (además de otras bibliotecas del sistema)

```
typedef struct {  
    int id cli;  
    string nif;  
    string nombre;  
    string telefono;  
} tCliente;
```



```
const int NCLI = 100;
```

```
typedef struct {  
    tCliente clientes[NCLI];  
    int cont;  
} tListaClientes;
```



```
void buscar(const tListaClientes &lista, string nif, tCliente  
            &cliente, bool &ok);
```

Luis Hernández Yáñez



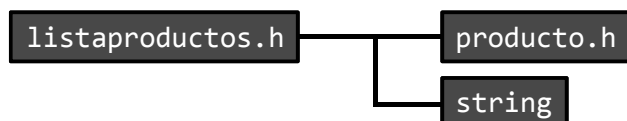
# Dependencias entre módulos

```
typedef struct {  
    int id_prod;  
    string codigo;  
    string nombre;  
    double precio;  
    int unidades;  
} tProducto;
```



```
const int NPROD = 200;
```

```
typedef struct {  
    tProducto productos[NPROD];  
    int cont;  
} tListaProductos;
```



```
void buscar(const tListaProductos &lista, string codigo, tProducto  
            &producto, bool &ok);
```

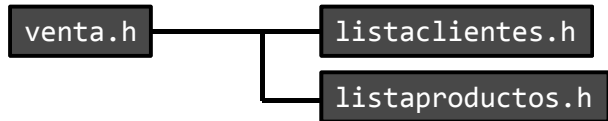
Luis Hernández Yáñez





# Dependencias entre módulos

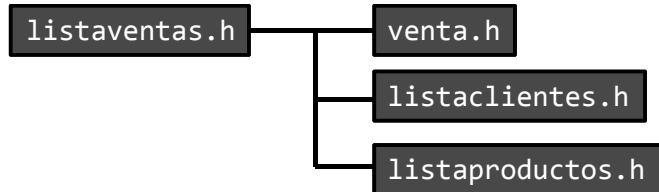
```
typedef struct {  
    int id;  
    int id_prod;  
    int id_cli;  
    int unidades;  
} tVenta;
```



```
void mostrar(tVenta venta, const tListaClientes &clientes,  
            const tListaProductos &productos);
```

```
const int NVENTAS = 3000;
```

```
typedef struct {  
    tVenta ventas[NVENTAS];  
    int cont;  
} tListaVentas;
```



```
double totalVentas(const tListaVentas &ventas, string nif,  
                  const tListaClientes &clientes,  
                  const tListaProductos &productos);
```

Luis Hernández Yáñez



# Protección frente a inclusiones múltiples

```
#ifndef cliente_h  
#define cliente_h
```

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

```
typedef struct {  
    int id_cli;  
    string nif;  
    string nombre;  
    string telefono;  
} tCliente;
```

```
tCliente nuevoCliente();  
bool operator<(tCliente opIzq, tCliente opDer); // Por NIF  
void mostrar(tCliente cliente);
```

```
#endif
```

Luis Hernández Yáñez








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.



## 9

# Punteros y memoria dinámica

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Direcciones de memoria y punteros	849
Operadores de punteros	854
Punteros y direcciones válidas	864
Punteros no inicializados	866
Un valor seguro: NULL	867
Copia y comparación de punteros	868
Tipos puntero	873
Punteros a estructuras	875
Punteros a constantes y punteros constantes	877
Punteros y paso de parámetros	879
Punteros y arrays	883
Memoria y datos del programa	886
Memoria dinámica	891
Punteros y datos dinámicos	895
Gestión de la memoria	907
Errores comunes	911
Arrays de datos dinámicos	916
Arrays dinámicos	928



## Direcciones de memoria y punteros

Luis Hernández Yáñez



Fundamentos de la programación: Punteros y memoria dinámica

Página 849

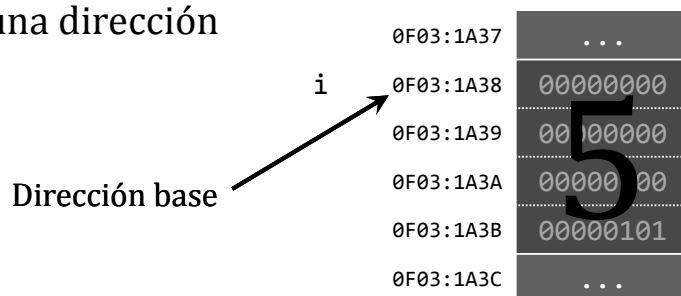


## Direcciones de memoria

### *Los datos en la memoria*

Todo dato se almacena en memoria:  
Varios bytes a partir de una dirección

```
int i = 5;
```



El dato ( $i$ ) se accede a partir de su *dirección base* (0F03:1A38)

Dirección de la primera celda de memoria utilizada por el dato

El tipo del dato (`int`) indica cuántos bytes (4) requiere el dato:

00000000 00000000 00000000 00000101  $\rightarrow$  5

(La codificación de los datos puede ser diferente; y la de las direcciones también)

Luis Hernández Yáñez



Fundamentos de la programación: Punteros y memoria dinámica

Página 850

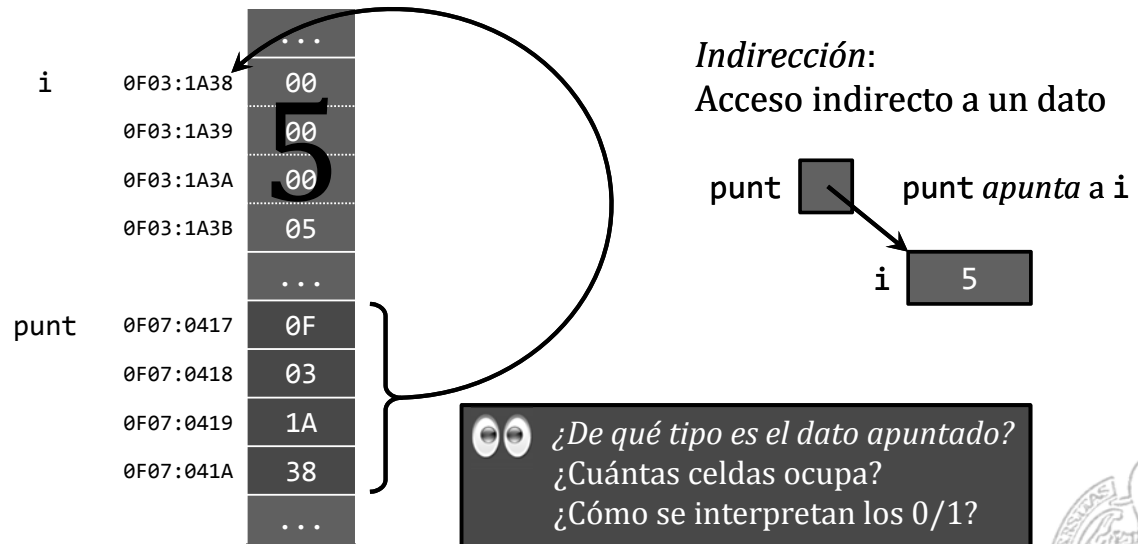


# Variables punteros

*Los punteros contienen direcciones de memoria*

Un *puntero* sirve para acceder a través de él a otro dato

El valor del puntero es la dirección de memoria base de otro dato



Luis Hernández Yáñez



# Punteros

*Los punteros contienen direcciones de memoria*

¿De qué tipo es el dato apuntado?

La variable a la que apunta un puntero será de un tipo concreto

¿Cuánto ocupa? ¿Cómo se interpreta?

El tipo de variable apuntado se establece al declarar el puntero:

*tipo \*nombre;*

El puntero *nombre* apuntará a una variable del *tipo* indicado

El asterisco (\*) indica que es un puntero a datos de ese tipo

```
int *punt; // punt inicialmente contiene una dirección  
           // que no es válida (no apunta a nada)
```

El puntero *punt* apuntará a una variable entera (int)

```
int i; // Dato entero vs. int *punt; // Puntero a entero
```

Luis Hernández Yáñez



# Punteros

---

## *Los punteros contienen direcciones de memoria*

Las variables puntero tampoco se inicializan automáticamente  
Al declararlas sin inicializador contienen direcciones no válidas

```
int *punt; // punt inicialmente contiene una dirección  
          // que no es válida (no apunta a nada)
```

Un puntero puede apuntar a cualquier dato de su tipo base

Un puntero no tiene por qué apuntar necesariamente a un dato  
(puede no apuntar a nada: valor NULL)

### *¿Para qué sirven los punteros?*

- ✓ Para implementar el paso de parámetros por referencia
- ✓ Para manejar datos dinámicos  
(Datos que se crean y destruyen durante la ejecución)
- ✓ Para implementar los arrays



---

# Fundamentos de la programación

## Operadores de punteros



# Operadores de punteros



## Obtener la dirección de memoria de ...

Operador monario y prefijo

& devuelve la dirección de memoria base del dato al que precede

```
int i;
```

```
cout << &i; // Muestra la dirección de memoria de i
```

Un puntero puede recibir la dirección de datos de su tipo base

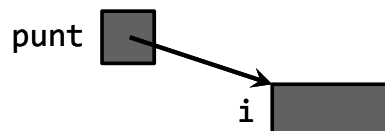
```
int i;
```

```
int *punt;
```

```
punt = &i; // punt contiene la dirección de i
```

Ahora punt ya contiene una dirección de memoria válida

punt *apunta* a (contiene la dirección de) la variable i (int)



# Operadores de punteros



## Obtener la dirección de memoria de ...

```
int i, j;
```

```
...
```

```
int *punt;
```

```
    i    0F03:1A38
```

```
        0F03:1A39
```

```
        0F03:1A3A
```

```
        0F03:1A3B
```

```
    j    0F03:1A3C
```

```
        0F03:1A3D
```

```
        0F03:1A3E
```

```
        0F03:1A3F
```

```
    punt 0F07:0417
```

```
        0F07:0418
```

```
        0F07:0419
```

```
        0F07:041A
```

```
    ...
```



## Obtener la dirección de memoria de ...

```
int i, j;  
...  
int *punt;  
...  
i = 5;
```



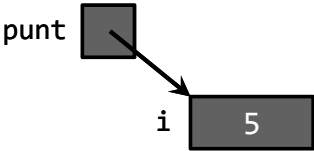
...	
i	0F03:1A38 00
	0F03:1A39 00
	0F03:1A3A 00
	0F03:1A3B 05
j	0F03:1A3C
	0F03:1A3D
	0F03:1A3E
	0F03:1A3F
...	
punt	0F07:0417
	0F07:0418
	0F07:0419
	0F07:041A
...	

Luis Hernández Yáñez



## Obtener la dirección de memoria de ...

```
int i, j;  
...  
int *punt;  
...  
i = 5;  
punt = &i;
```



...	
i	0F03:1A38 00
	0F03:1A39 00
	0F03:1A3A 00
	0F03:1A3B 05
j	0F03:1A3C
	0F03:1A3D
	0F03:1A3E
	0F03:1A3F
...	
punt	0F07:0417 0F
	0F07:0418 03
	0F07:0419 1A
	0F07:041A 38
...	

Luis Hernández Yáñez





# Operadores de punteros

\*

*Obtener lo que hay en la dirección ...*

Operador monario y prefijo

\* accede a lo que hay en la dirección de memoria a la que precede

Permite acceder a un dato a través un puntero que lo apunte:

```
punt = &i;
```

```
cout << *punt; // Muestra lo que hay en la dirección punt
```

\*punt: lo que hay en la dirección que contiene el puntero punt

punt contiene la dirección de memoria de la variable i

\*punt accede al contenido de esa variable i

*Acceso indirecto* al valor de i

Luis Hernández Yáñez



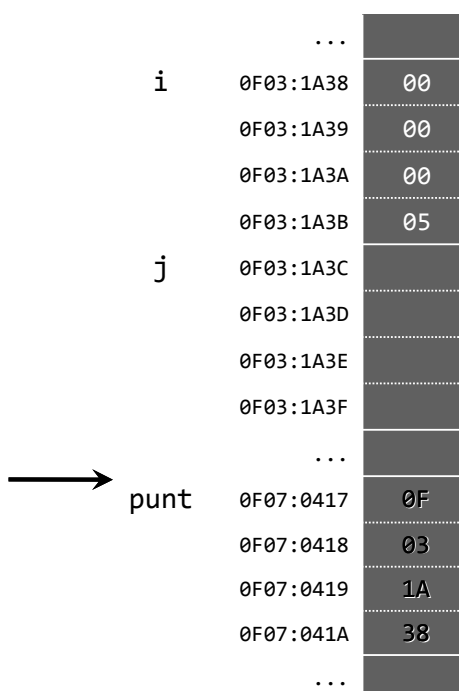
# Operadores de punteros

\*

*Obtener lo que hay en la dirección ...*

```
int i, j;  
...  
int *punt;  
...  
i = 5;  
punt = &i;  
j = *punt;
```

punt:



Luis Hernández Yáñez



# Operadores de punteros

\*

*Obtener lo que hay en la dirección ...*

```
int i, j;  
...  
int *punt;  
...  
i = 5;  
punt = &i;  
j = *punt;
```

Direccionamiento indirecto (indirección)  
Se accede al dato **i** de forma indirecta



...		
i	0F03:1A38	00
	0F03:1A39	00
	0F03:1A3A	00
	0F03:1A3B	05
j	0F03:1A3C	
	0F03:1A3D	
	0F03:1A3E	
	0F03:1A3F	
...		
punt	0F07:0417	0F
	0F07:0418	03
	0F07:0419	1A
	0F07:041A	38
...		

\*punt:

Luis Hernández Yáñez



# Operadores de punteros

\*

*Obtener lo que hay en la dirección ...*

```
int i, j;  
...  
int *punt;  
...  
i = 5;  
punt = &i;  
j = *punt;
```



...		
i	0F03:1A38	00
	0F03:1A39	00
	0F03:1A3A	00
	0F03:1A3B	05
j	0F03:1A3C	00
	0F03:1A3D	00
	0F03:1A3E	00
	0F03:1A3F	05
...		
punt	0F07:0417	0F
	0F07:0418	03
	0F07:0419	1A
	0F07:041A	38
...		

Luis Hernández Yáñez



## Ejemplo de uso de punteros

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

int main() {
    int i = 5;
    int j = 13;
    int *punt;
    punt = &i;
    cout << *punt << endl; // Muestra el valor de i
    punt = &j;
    cout << *punt << endl; // Ahora muestra el valor de j
    int *otro = &i;
    cout << *otro + *punt << endl; // i + j
    int k = *punt;
    cout << k << endl; // Mismo valor que j

    return 0;
}
```

```
5
13
18
13
```



Luis Hernández Yáñez



# Fundamentos de la programación

## Punteros y direcciones válidas

Luis Hernández Yáñez



# Punteros y direcciones válidas

---

*Todo puntero ha de tener una dirección válida*

Un puntero sólo debe ser utilizado si tiene una dirección válida

Un puntero NO contiene una dirección válida tras ser definido

Un puntero obtiene una dirección válida:

- ✓ Asignando la dirección de otro dato (operador &)
- ✓ Asignando otro puntero (mismo tipo base) que ya sea válido
- ✓ Asignando el valor NULL (puntero nulo, no apunta a nada)

```
int i;
int *q; // q no tiene aún una dirección válida
int *p = &i; // p toma una dirección válida
q = p; // ahora q ya tiene una dirección válida
q = NULL; // otra dirección válida para q
```



# Punteros no inicializados

---

*Punteros que apuntan a saber qué...*

Un puntero no inicializado contiene una dirección desconocida

```
int *punt; // No inicializado
```

```
*punt = 12; // ¿A qué dato se está asignando el valor?
```

*¿Dirección de la zona de datos del programa?*

¡Podemos modificar inadvertidamente un dato del programa!

*¿Dirección de la zona de código del programa?*

¡Podemos modificar el código del propio programa!

*¿Dirección de la zona de código del sistema operativo?*

¡Podemos modificar el código del propio S.O.!

→ Consecuencias imprevisibles (*cuelgue*)

(Los S.O. modernos protegen bien la memoria)



**¡SALVAJES!**



# Un valor seguro: NULL

---

## *Punteros que no apuntan a nada*

Inicializando los punteros a NULL podemos detectar errores:

```
int *punt = NULL;
```

```
...
```

```
*punt = 13;
```

punt 

punt ha sido inicializado a NULL: ¡No apunta a nada!

Si no apunta a nada, ¿¿¿qué significa \*punt??? No tiene sentido

→ ERROR: ¡Acceso a un dato a través de un puntero nulo!

Error de ejecución, lo que ciertamente no es bueno

Pero sabemos cuál ha sido el problema, lo que es mucho

Sabemos dónde y qué buscar para depurar



---

# Fundamentos de la programación

## Copia y comparación de punteros

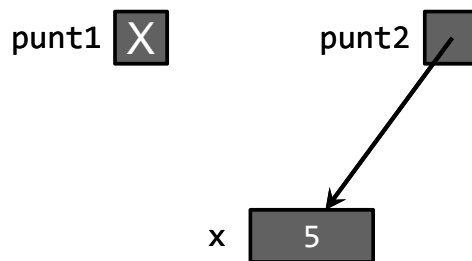


# Copia de punteros

## Apuntando al mismo dato

Al copiar un puntero en otro, ambos apuntarán al mismo dato:

```
int x = 5;  
int *punt1 = NULL; // punt1 no apunta a nada  
int *punt2 = &x; // punt2 apunta a la variable x
```



Luis Hernández Yáñez

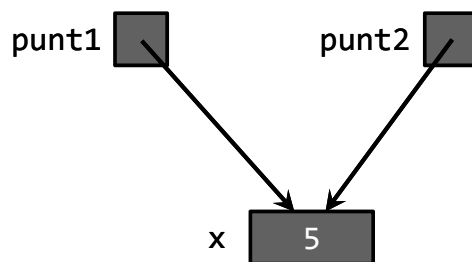


# Copia de punteros

## Apuntando al mismo dato

Al copiar un puntero en otro, ambos apuntarán al mismo dato:

```
int x = 5;  
int *punt1 = NULL; // punt1 no apunta a nada  
int *punt2 = &x; // punt2 apunta a la variable x  
punt1 = punt2; // ambos apuntan a la variable x
```



Luis Hernández Yáñez

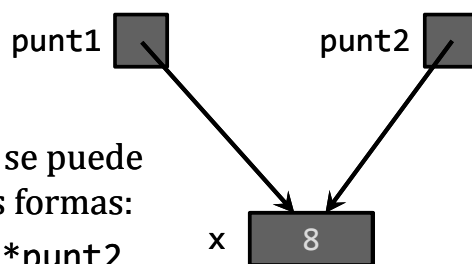


# Copia de punteros

## Apuntando al mismo dato

Al copiar un puntero en otro, ambos apuntarán al mismo dato:

```
int x = 5;  
int *punt1 = NULL; // punt1 no apunta a nada  
int *punt2 = &x; // punt2 apunta a la variable x  
punt1 = punt2; // ambos apuntan a la variable x  
*punt1 = 8;
```



Al dato x ahora se puede acceder de tres formas:

x \*punt1 \*punt2



# Comparación de punteros

## ¿Apuntan al mismo dato?

Operadores relacionales == y !=:

```
int x = 5;  
int *punt1 = NULL;  
int *punt2 = &x;  
...  
if (punt1 == punt2) {  
    cout << "Apuntan al mismo dato" << endl;  
}  
else {  
    cout << "No apuntan al mismo dato" << endl;  
}
```

Sólo se pueden comparar punteros con el mismo tipo base



## Tipos puntero



## Tipos puntero

tipos.cpp

### *Declaración de tipos puntero*

Declaramos tipos para los punteros con distintos tipos base:

```
typedef int *tIntPtr;  
typedef char *tCharPtr;  
typedef double *tDoublePtr;  
int entero = 5;  
tIntPtr puntI = &entero;  
char caracter = 'C';  
tCharPtr puntC = &caracter;  
double real = 5.23;  
tDoublePtr puntD = &real;  
cout << *puntI << " " << *puntC << " " << *puntD << endl;
```

Con *\*puntero* podemos hacer lo que con otros datos del tipo base





# Punteros a estructuras

## Acceso a estructuras a través de punteros

Los punteros pueden apuntar también a estructuras:

```
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;
tRegistro registro;
typedef tRegistro *tRegistroPtr;
tRegistroPtr puntero = &registro;
```

Operador flecha (->):

Acceso a los campos a través de un puntero sin usar el operador \*

```
puntero->codigo    puntero->nombre    puntero->sueldo
puntero->... ≡ (*puntero)....
```



# Punteros a estructuras

structPtr.cpp

## Acceso a estructuras a través de punteros

```
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;
tRegistro registro;
typedef tRegistro *tRegistroPtr;
tRegistroPtr puntero = &registro;
registro.codigo = 12345;
registro.nombre = "Javier";
registro.sueldo = 95000;
cout << puntero->codigo << " " << puntero->nombre
     << " " << puntero->sueldo << endl;
```

$\text{puntero->codigo} \equiv (*\text{puntero}).\text{codigo} \neq \underbrace{*}\text{puntero}.\text{codigo}$

puntero sería una estructura con campo codigo de tipo puntero



# Punteros y el modificador const

---

## *Punteros a constantes y punteros constantes*

El efecto del modificador de acceso const depende de su sitio:

const tipo \*puntero;      Puntero a una constante

tipo \*const puntero;      Puntero constante

Punteros a constantes:

```
typedef const int *tIntCtePtr; // Puntero a constante
```

```
int entero1 = 5, entero2 = 13;
```

```
tIntCtePtr punt_a_cte = &entero1;
```

```
(*punt_a_cte)++; // ERROR: ¡Dato no modificable!
```

```
punt_a_cte = &entero2; // OK: El puntero no es cte.
```



# Punteros y el modificador const

---

constPtr.cpp

## *Punteros a constantes y punteros constantes*

El efecto del modificador de acceso const depende de su sitio:

const tipo \*puntero;      Puntero a una constante

tipo \*const puntero;      Puntero constante

Punteros constantes:

```
typedef int *const tIntPtrCte; // Puntero constante
```

```
int entero1 = 5, entero2 = 13;
```

```
tIntPtrCte punt_cte = &entero1;
```

```
(*punt_cte)++; // OK: El puntero no apunta a cte.
```

```
punt_cte = &entero2; // ERROR: ¡Puntero constante!
```



## Punteros y paso de parámetros



## Punteros y paso de parámetros

param.cpp

### *Paso de parámetros por referencia o variable*

En el lenguaje C no hay mecanismo de paso por referencia (&)

Sólo se pueden pasar parámetros por valor

¿Cómo se simula el paso por referencia? Por medio de punteros:

```
void incrementa(int *punt);
```

```
void incrementa(int *punt) {  
    (*punt)++;  
}
```

```
...  
int entero = 5;  
incrementa(&entero);  
cout << entero << endl;
```

Mostrará 6 en la consola

Paso por valor:

El argumento (el puntero) no cambia

Aquello a lo que apunta (el entero)

Sí puede cambiar



# Punteros y paso de parámetros



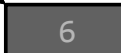
## *Paso de parámetros por referencia o variable*

```
int entero = 5;  
incrementa(&entero);
```

entero 

punt recibe la dirección de entero

```
void incrementa(int *punt) {  
    (*punt)++;  
}
```

punt   entero 

```
cout << entero << endl;
```

entero 

Luis Hernández Yáñez



# Punteros y paso de parámetros

## *Paso de parámetros por referencia o variable*

¿Cuál es el equivalente en C a este prototipo de C++?

```
void foo(int &param1, double &param2, char &param3);
```

Prototipo equivalente:

```
void foo(int *param1, double *param2, char *param3);
```

```
void foo(int *param1, double *param2, char *param3) {  
    // Al primer argumento se accede con *param1  
    // Al segundo argumento se accede con *param2  
    // Al tercer argumento se accede con *param3  
}
```

¿Cómo se llamaría?

```
int entero; double real; char caracter;  
//...  
foo(&entero, &real, &caracter);
```

Luis Hernández Yáñez



## Punteros y arrays



## Punteros y arrays

---

### *Una íntima relación*

Variable array  $\equiv$  Puntero al primer elemento del array


Así, si tenemos:

```
int dias[12] =  
{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Entonces:

```
cout << *dias << endl;
```

Muestra 31 en la consola, el primer elemento del array

 ¡Un nombre de array es un puntero constante!

Siempre apunta al primer elemento (no se puede modificar)

Acceso a los elementos del array:

Por índice o con aritmética de punteros (Anexo)



# Punteros y paso de parámetros arrays

---

## *Paso de arrays a subprogramas*

*¡Esto explica por qué no usamos & con los parámetros array!*

El nombre del array es un puntero: ya es un paso por referencia

Prototipos equivalentes para parámetros array:

```
const int N = ...;
void cuadrado(int arr[N]);
void cuadrado(int arr[], int size); // Array no delimitado
void cuadrado(int *arr, int size); // Puntero
```

Arrays no delimitados y punteros: se necesita la dimensión

Elementos: se acceden con índice (`arr[i]`) o con puntero (`*arr`)

Una función sólo puede devolver un array en forma de puntero:

```
intPtr inicializar();
```



# Fundamentos de la programación

---

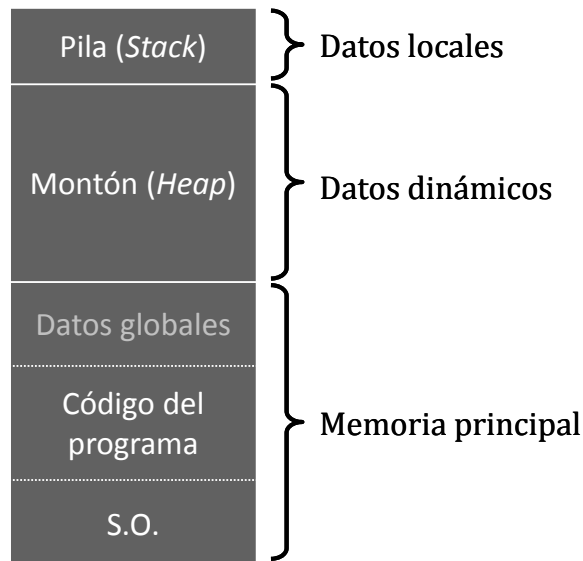
## Memoria y datos del programa



# Memoria y datos del programa

## Regiones de la memoria

El sistema operativo distingue varias regiones en la memoria:



Luis Hernández Yáñez

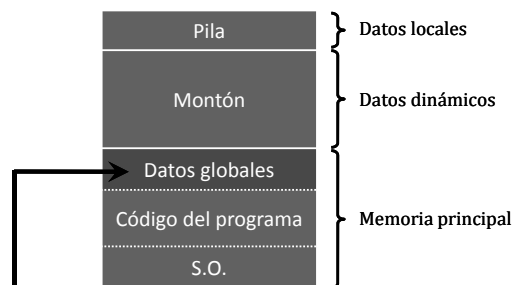


# Memoria y datos del programa

## Memoria principal

Datos globales del programa:  
Declarados fuera  
de los subprogramas

```
typedef struct {  
    ...  
} tRegistro;  
const int N = 1000;  
typedef tRegistro tArray[N];  
typedef struct {  
    tArray registros;  
    int cont;  
} tLista;  
  
int main() {  
    ...  
}
```



Luis Hernández Yáñez



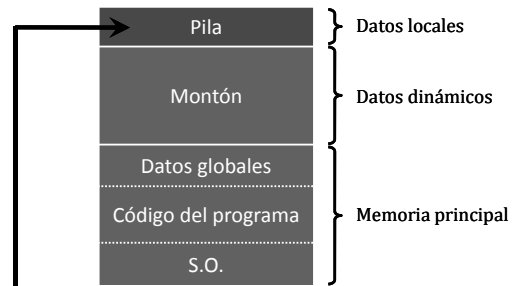
# Memoria y datos del programa

## La pila (stack)

Datos locales de subprogramas:  
Parámetros por valor  
y variables locales

```
void func(tLista lista, double &total)
{
    tLista aux;
    int i;
    ...
}
```

Y los punteros temporales  
que apuntan a los argumentos  
de los parámetros por referencia



&resultado  
func(lista, resultado)

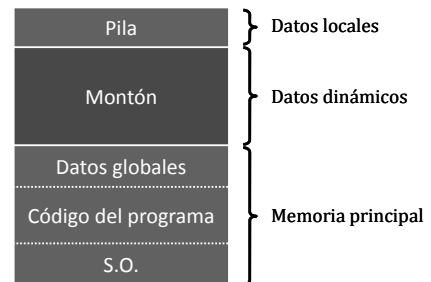


# Memoria y datos del programa

## El montón (heap)

### Datos dinámicos

Datos que se crean y se destruyen  
durante la ejecución del programa,  
a medida que se necesita



Sistema de gestión de memoria dinámica (SGMD)

Cuando se necesita memoria para una variable se solicita

El SGMD reserva espacio y devuelve la dirección base

Cuando ya no se necesita más la variable, se destruye

Se libera la memoria y el SGMD cuenta de nuevo con ella





## Memoria dinámica



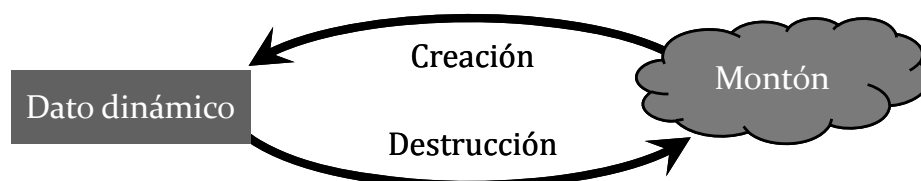
## Memoria dinámica

---

### *Datos dinámicos*

Se crean y se destruyen durante la ejecución del programa

Se les asigna memoria del montón



¿Por qué utilizar memoria dinámica?

- ✓ Almacén de memoria muy grande: datos o listas de datos que no caben en memoria principal pueden caber en el montón
- ✓ El programa ajusta el uso de la memoria a las necesidades de cada momento: ni le falta ni la desperdicia



# Datos y asignación de memoria

---

## *¿Cuándo se asigna memoria a los datos?*

- ✓ Datos globales  
En memoria principal al comenzar la ejecución del programa  
Existen durante toda la ejecución del programa
- ✓ Datos locales de un subprograma  
En la pila al ejecutarse el subprograma  
Existen sólo durante la ejecución de su subprograma
- ✓ Datos dinámicos  
En el montón cuando el programa lo solicita  
Existen *a voluntad* del programa



# Datos estáticos frente a datos dinámicos

---

## *Datos estáticos*

- ✓ Datos declarados como de un tipo concreto:  
`int i;`
- ✓ Se acceden directamente a través del identificador:  
`cout << i;`

## *Datos dinámicos*

- ✓ Datos accedidos a través de su dirección de memoria  
Esa dirección de memoria debe estar en algún puntero  
Los punteros son la base del SGMD

Los datos estáticos también se pueden acceder a través de punteros

```
int *p = &i;
```



## Punteros y datos dinámicos



## Creación de datos dinámicos

---

*El operador new*

Devuelve NULL si no queda memoria suficiente

`new tipo` Reserva memoria del montón para una variable del *tipo* y devuelve la primera dirección de memoria utilizada, que debe ser asignada a un puntero

```
int *p; // Todavía sin una dirección válida
p = new int; // Ya tiene una dirección válida
*p = 12;
```

La variable dinámica se accede exclusivamente por punteros

No tiene identificador asociado

```
int i; // i es una variable estática
int *p1, *p2;
p1 = &i; // Puntero que da acceso a la variable
        // estática i (accesible con i o con *p1)
p2 = new int; // Puntero que da acceso a una variable
              // dinámica (accesible sólo a través de p2)
```



## Inicialización con el operador new

El operador new admite un valor inicial para el dato creado:

```
int *p;  
p = new int(12);
```

Se crea la variable, de tipo int, y se inicializa con el valor 12

```
#include <iostream>  
using namespace std;  
#include "registro.h"
```

```
int main() {  
    tRegistro reg;  
    reg = nuevo();  
    tRegistro *punt = new tRegistro(reg);  
    mostrar(*punt);  
    ...  
}
```



# Eliminación de datos dinámicos

## El operador delete

`delete puntero;` Devuelve al montón la memoria usada por la variable dinámica apuntada por *puntero*

```
int *p;  
p = new int;  
*p = 12;  
...  
delete p; // Ya no se necesita el entero apuntado por p
```

¡El puntero deja de contener una dirección válida!



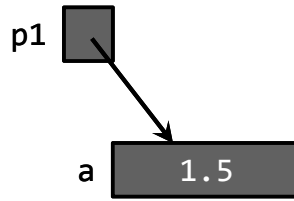
# Ejemplo de variables dinámicas

dinamicas.cpp

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

int main() {
    → double a = 1.5;
      double *p1, *p2, *p3;
      p1 = &a;
      p2 = new double;
      *p2 = *p1;
      p3 = new double;
      *p3 = 123.45;
      cout << *p1 << endl;
      cout << *p2 << endl;
      cout << *p3 << endl;
      delete p2;
      delete p3;

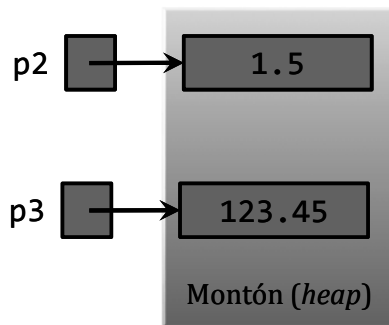
      return 0;
}
```



Identificadores:

4

(a, p1, p2, p3)



Variables:

6

(+ \*p2 y \*p3)

Luis Hernández Yáñez



Fundamentos de la programación: Punteros y memoria dinámica

Página 899



# Ejemplo de variables dinámicas

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

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
```



Luis Hernández Yáñez



Fundamentos de la programación: Punteros y memoria dinámica

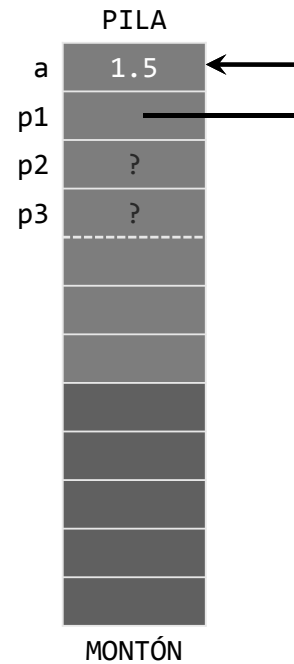
Página 900



# Ejemplo de variables dinámicas

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

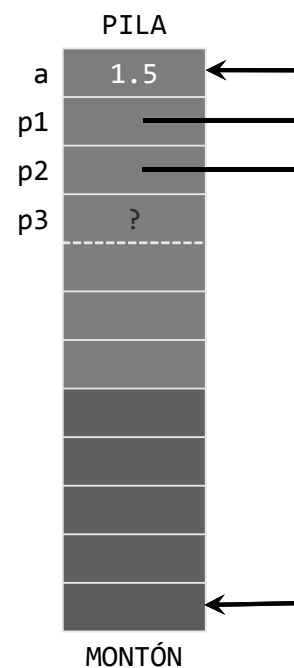
int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
```



# Ejemplo de variables dinámicas

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

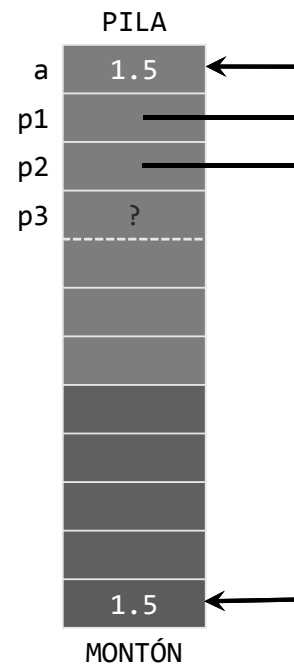
int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
```



# Ejemplo de variables dinámicas

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

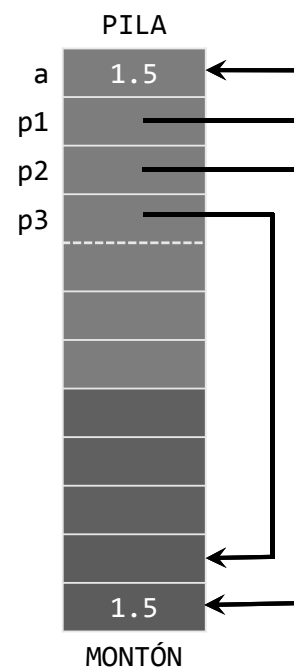
int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
```



# Ejemplo de variables dinámicas

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

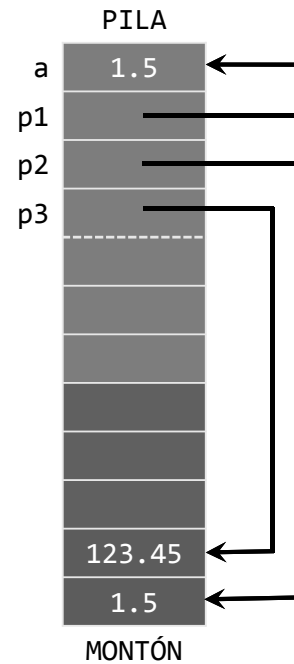
int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
```



# Ejemplo de variables dinámicas

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

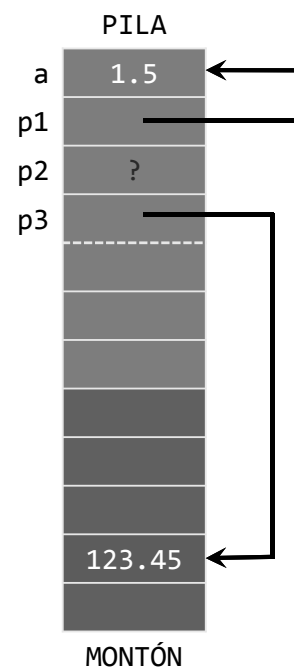
int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
    *p3 = 123.45;
```



# Ejemplo de variables dinámicas

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

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
    *p3 = 123.45;
    cout << *p1 << endl;
    cout << *p2 << endl;
    cout << *p3 << endl;
    delete p2;
```

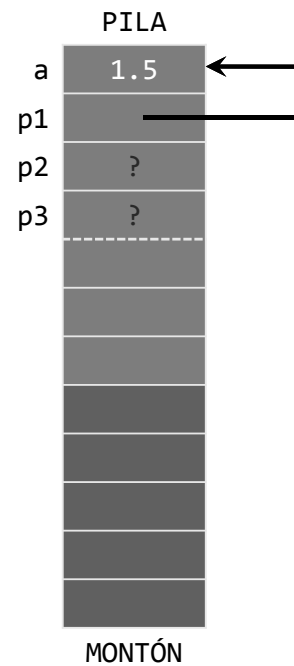




# Ejemplo de variables dinámicas

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

int main() {
    double a = 1.5;
    double *p1, *p2, *p3;
    p1 = &a;
    p2 = new double;
    *p2 = *p1;
    p3 = new double;
    *p3 = 123.45;
    cout << *p1 << endl;
    cout << *p2 << endl;
    cout << *p3 << endl;
    delete p2;
    delete p3;
}
```



# Fundamentos de la programación

## Gestión de la memoria



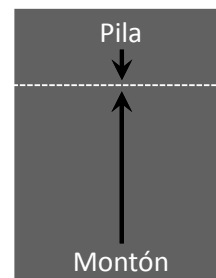
# Errores de asignación de memoria

La memoria se reparte entre la pila y el montón

Crecen en direcciones opuestas

Al llamar a subprogramas la pila crece

Al crear datos dinámicos el montón crece



## *Colisión pila-montón*

Los límites de ambas regiones se encuentran

Se agota la memoria

## *Desbordamiento de la pila*

La pila suele tener un tamaño máximo establecido

Si se sobrepasa se agota la pila



# Gestión de la memoria dinámica

## *Gestión del montón*

Sistema de Gestión de Memoria Dinámica (SGMD)

Gestiona la asignación de memoria a los datos dinámicos

Localiza secciones adecuadas y sigue la pista de lo disponible

No dispone de un *recolector de basura*, como el lenguaje Java

*¡Hay que devolver toda la memoria solicitada!*

Deben ejecutarse tantos `delete` como `new` se hayan ejecutado

La memoria disponible en el montón debe ser exactamente la misma antes y después de la ejecución del programa

Y todo dato dinámico debe tener algún acceso (puntero)

Es un grave error *perder* un dato en el montón



## Errores comunes



## Mal uso de la memoria dinámica I

---

### *Olvido de destrucción de un dato dinámico*

```
...
int main() {
    tRegistro *p;
    p = new tRegistro;
    *p = nuevo();
    mostrar(*p);
    return 0;
}
```



Falta delete p;

G++ no indicará ningún error y el programa parecerá terminar correctamente, pero dejará memoria desperdiciada

Visual C++ sí comprueba el uso de la memoria dinámica y nos avisa si dejamos memoria sin liberar

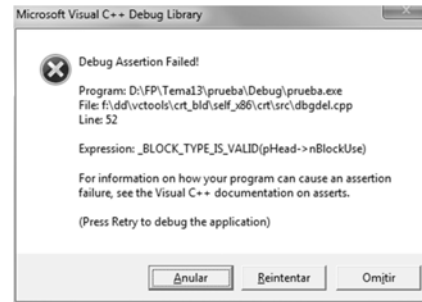
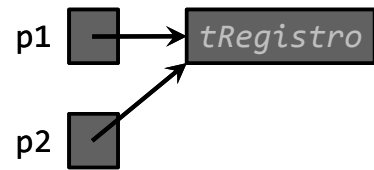


# Mal uso de la memoria dinámica II

## Intento de destrucción de un dato inexistente

```
...
int main() {
    tRegistro *p1 = new tRegistro;
    *p1 = nuevo();
    mostrar(*p1);
    tRegistro *p2;
    p2 = p1;
    mostrar(*p2);
    delete p1;
    delete p2;

    return 0;
}
```



👁️ Sólo se ha creado una variable



# Mal uso de la memoria dinámica III

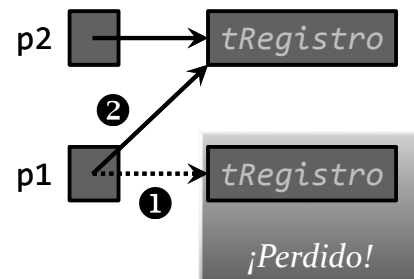
## Pérdida de un dato dinámico

```
...
int main() {
    tRegistro *p1, *p2;
    p1 = new tRegistro(nuevo()); ①
    p2 = new tRegistro(nuevo());

    mostrar(*p1);
    p1 = p2; ②
    mostrar(*p1);

    delete p1;
    delete p2;

    return 0;
}
```



👁️ Se pierde un dato en el montón  
Se intenta eliminar un dato ya eliminado



# Mal uso de la memoria dinámica IV

---

## *Intento de acceso a un dato tras su eliminación*

```
...
int main() {
    tRegistro *p;
    p = new tRegistro(nuevo());

    mostrar(*p);
    delete p;
    ...
    mostrar(*p); ←
    return 0;
}
```



p ha dejado de apuntar al dato dinámico destruido  
→ Acceso a memoria inexistente



# Fundamentos de la programación

---

## Arrays de datos dinámicos



# Arrays de datos dinámicos

## Arrays de punteros a datos dinámicos

```
typedef struct {  
    int codigo;  
    string nombre;  
    double valor;  
} tRegistro;  
typedef tRegistro *tRegPtr;
```

Los punteros ocupan muy poco en memoria  
Los datos a los que apunten estarán en el montón

```
const int N = 1000;  
// Array de punteros a registros:  
typedef tRegPtr tArray[N];  
typedef struct {  
    tArray registros;  
    int cont;  
} tLista;
```

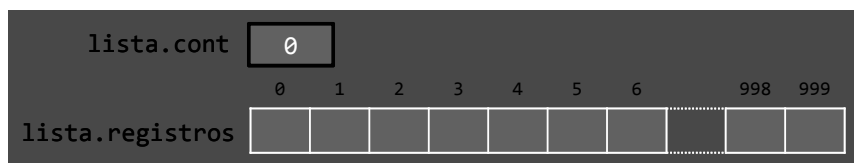
Se crean a medida que se insertan  
Se destruyen a medida que se eliminan

Luis Hernández Yáñez



# Arrays de datos dinámicos

```
tLista lista;  
lista.cont = 0;
```

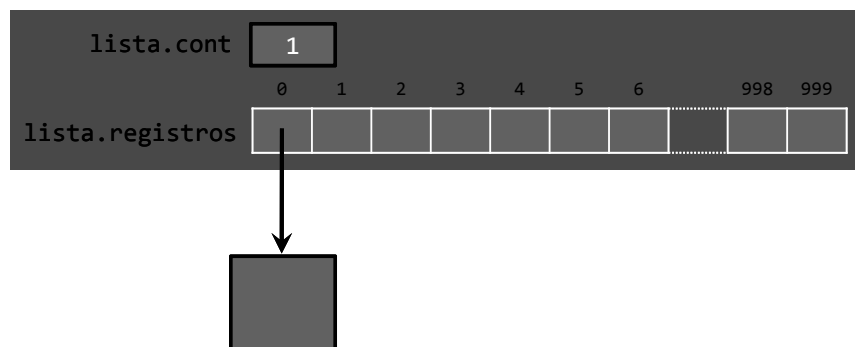


Luis Hernández Yáñez



# Arrays de datos dinámicos

```
tLista lista;  
lista.cont = 0;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;
```

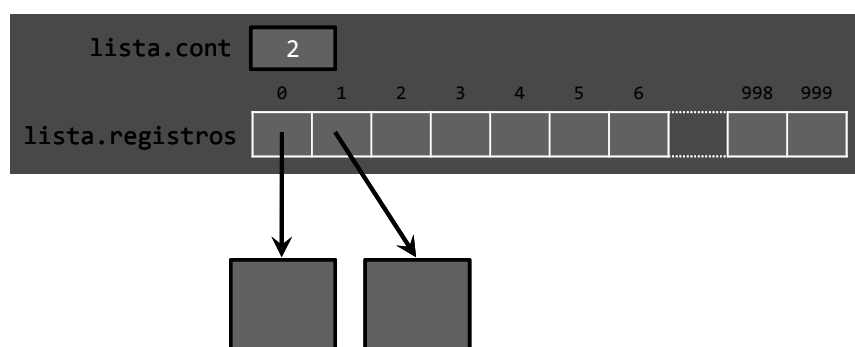


Luis Hernández Yáñez



# Arrays de datos dinámicos

```
tLista lista;  
lista.cont = 0;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;
```

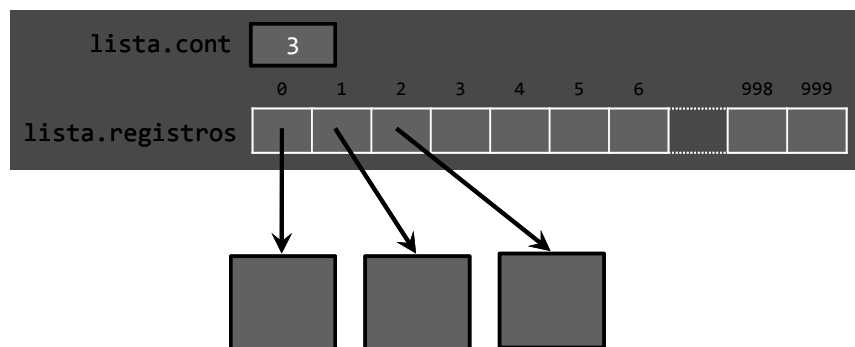


Luis Hernández Yáñez



# Arrays de datos dinámicos

```
tLista lista;  
lista.cont = 0;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;
```

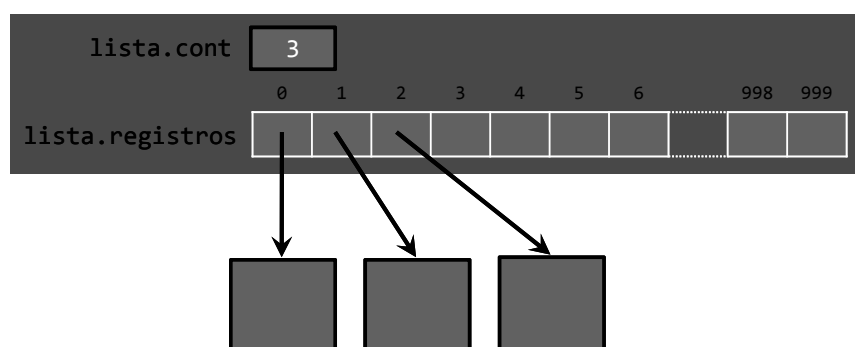


Luis Hernández Yáñez



# Arrays de datos dinámicos

Los registros se acceden a través de los punteros (operador `->`):  
`cout << lista.registros[0]->nombre;`



Luis Hernández Yáñez

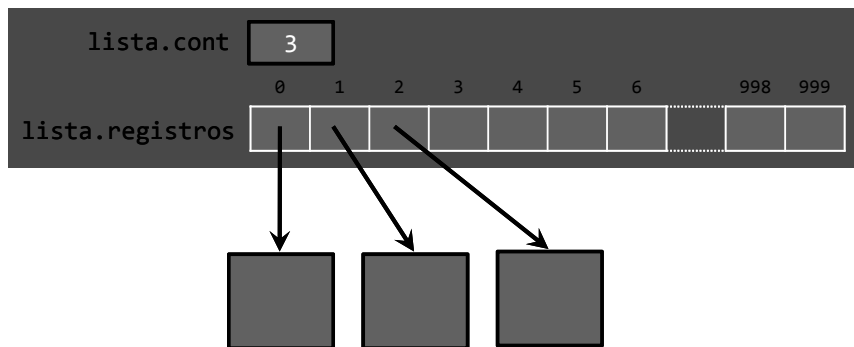




# Arrays de datos dinámicos

No hay que olvidarse de devolver la memoria al montón:

```
for (int i = 0; i < lista.cont; i++) {  
    delete lista.registros[i];  
}
```



Luis Hernández Yáñez



# Arrays de datos dinámicos

lista.h

```
#ifndef lista_h  
#define lista_h  
#include "registro.h"  
  
const int N = 1000;  
const string BD = "bd.dat";  
typedef tRegPtr tArray[N];  
typedef struct {  
    tArray registros;  
    int cont;  
} tLista;  
  
void mostrar(const tLista &lista);  
void insertar(tLista &lista, tRegistro registro, bool &ok);  
void eliminar(tLista &lista, int code, bool &ok);  
int buscar(const tLista &lista, int code);  
void cargar(tLista &lista, bool &ok);  
void guardar(const tLista &lista);  
void destruir(tLista &lista);  
  
#endif
```

registro.h con el tipo puntero:  
typedef tRegistro \*tRegPtr;

Luis Hernández Yáñez



```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false;
    }
    else {
        lista.registros[lista.cont] = new tRegistro(registro);
        lista.cont++;
    }
}

void eliminar(tLista &lista, int code, bool &ok) {
    ok = true;
    int ind = buscar(lista, code);
    if (ind == -1) {
        ok = false;
    }
    else {
        delete lista.registros[ind];
        for (int i = ind + 1; i < lista.cont; i++) {
            lista.registros[i - 1] = lista.registros[i];
        }
        lista.cont--;
    }
}
```

Luis Hernández Yáñez



# Arrays de datos dinámicos

```
int buscar(const tLista &lista, int code) {
    // Devuelve el índice o -1 si no se ha encontrado
    int ind = 0;
    bool encontrado = false;
    while ((ind < lista.cont) && !encontrado) {
        if (lista.registros[ind]->codigo == code) {
            encontrado = true;
        }
        else {
            ind++;
        }
    }
    if (!encontrado) {
        ind = -1;
    }
    return ind;
}

void destruir(tLista &lista) {
    for (int i = 0; i < lista.cont; i++) {
        delete lista.registros[i];
    }
    lista.cont = 0;
}

...

```

Luis Hernández Yáñez



```
#include <iostream>
using namespace std;
#include "registro.h"
#include "lista.h"

int main() {
    tLista lista;
    bool ok;
    cargar(lista, ok);
    if (ok) {
        mostrar(lista);
        destruir(lista);
    }

    return 0;
}
```

Elementos de la lista:

```
-----
12345 - Disco duro - 123.59 euros
324356 - Placa base core i7 - 234.50 euros
2121 - Multipuerto USB - 15.00 euros
54354 - Disco externo 500 Gb - 95.00 euros
112341 - Procesador AMD - 132.95 euros
66678325 - Marco digital 2 Gb - 78.99 euros
600673 - Monitor 22" Nisu - 154.50 euros
```



## Fundamentos de la programación

# Arrays dinámicos



# Arrays dinámicos

---

## *Creación y destrucción de arrays dinámicos*

Array dinámico: array que se ubica en la memoria dinámica

Creación de un array dinámico:

```
tipo *puntero = new tipo[dimensión];
```

```
int *p = new int[10];
```

Crea un array de 10 int en memoria dinámica

Los elementos se acceden a través del puntero: `p[i]`

Destrucción del array:

```
delete [] p;
```



# Arrays dinámicos

---

```
#include <iostream>
using namespace std;
const int N = 10;

int main() {
    int *p = new int[N];
    for (int i = 0; i < N; i++) {
        p[i] = i;
    }
    for (int i = 0; i < N; i++) {
        cout << p[i] << endl;
    }
    delete [] p;

    return 0;
}
```

¡No olvides destruir el array dinámico!



# Ejemplo de array dinámico

listaAD.h

```
...
#include "registro.h"

const int N = 1000;

// Lista: array dinámico (puntero) y contador
typedef struct {
    tRegPtr registros;
    int cont;
} tLista;

...
```

Luis Hernández Yáñez



# Ejemplo de array dinámico

listaAD.cpp

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false;
    }
    else {
        lista.registros[lista.cont] = registro;
        lista.cont++;
    }
}
```

No usamos new

Se han creado todo el array al cargar

```
void eliminar(tLista &lista, int code, bool &ok) {
    ok = true;
    int ind = buscar(lista, code);
    if (ind == -1) {
        ok = false;
    }
    else {
        for (int i = ind + 1; i < lista.cont; i++) {
            lista.registros[i - 1] = lista.registros[i];
        }
        lista.cont--;
    }
} ...
```

No usamos delete

Se destruye todo el array al final

Luis Hernández Yáñez



# Ejemplo de array dinámico

```
int buscar(tLista lista, int code) {
    int ind = 0;
    bool encontrado = false;
    while ((ind < lista.cont) && !encontrado) {
        if (lista.registros[ind].codigo == code) {
            encontrado = true;
        }
        else {
            ind++;
        }
    }
    if (!encontrado) {
        ind = -1;
    }
    return ind;
}

void destruir(tLista &lista) {
    delete [] lista.registros;
    lista.cont = 0;
}
...
```

Luis Hernández Yáñez



# Ejemplo de array dinámico

```
void cargar(tLista &lista, bool &ok) {
    ifstream archivo;
    char aux;
    ok = true;
    archivo.open(BD.c_str());
    if (!archivo.is_open()) {
        ok = false;
    }
    else {
        tRegistro registro;
        lista.cont = 0;
        lista.registros = new tRegistro[N];
        archivo >> registro.codigo;
        while ((registro.codigo != -1) && (lista.cont < N)) {
            archivo >> registro.valor;
            archivo.get(aux); // Saltamos el espacio
            getline(archivo, registro.nombre);
            lista.registros[lista.cont] = registro;
            lista.cont++;
            archivo >> registro.codigo;
        }
        archivo.close();
    }
}
```

Se crean todos a la vez



Luis Hernández Yáñez



Mismo programa principal que el del array de datos dinámicos  
Pero incluyendo listaAD.h, en lugar de lista.h

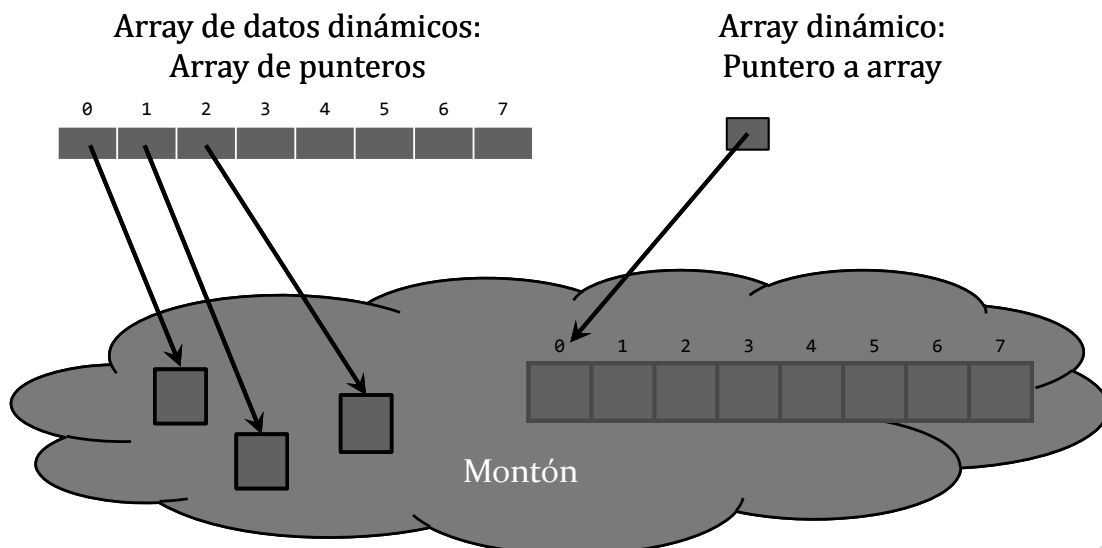
```
Elementos de la lista:
-----
12345 - Disco duro           - 123.59 euros
324356 - Placa base core i7   - 234.50 euros
2121 - Multipuerto USB        - 15.00 euros
54354 - Disco externo 500 Gb  - 95.00 euros
112341 - Procesador AMD        - 132.95 euros
66678325 - Marco digital 2 Gb - 78.99 euros
600673 - Monitor 22" Nisu     - 154.50 euros
```



## Arrays dinámicos vs. arrays de dinámicos

Array de datos dinámicos: Array de punteros a datos dinámicos

Array dinámico: Puntero a array en memoria dinámica








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

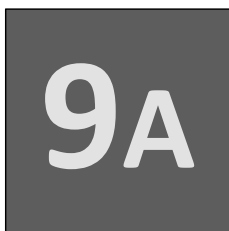
Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.







ANEXO

## Punteros y memoria dinámica

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Aritmética de punteros	940
Recorrido de arrays con punteros	953
Referencias	962
Listas enlazadas	964



## Aritmética de punteros



## Aritmética de punteros

---

### *Operaciones aritméticas con punteros*

La aritmética de punteros es una aritmética un tanto especial...

Trabaja tomando como unidad de cálculo el tamaño del tipo base

```
int dias[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

```
typedef int* tIntPtr;
```

```
tIntPtr punt = dias;
```

punt empieza apuntando al primer elemento del array:

```
cout << *punt << endl; // Muestra 31 (primer elemento)
```

```
punt++;
```

punt++ hace que punt pase a apuntar al siguiente elemento

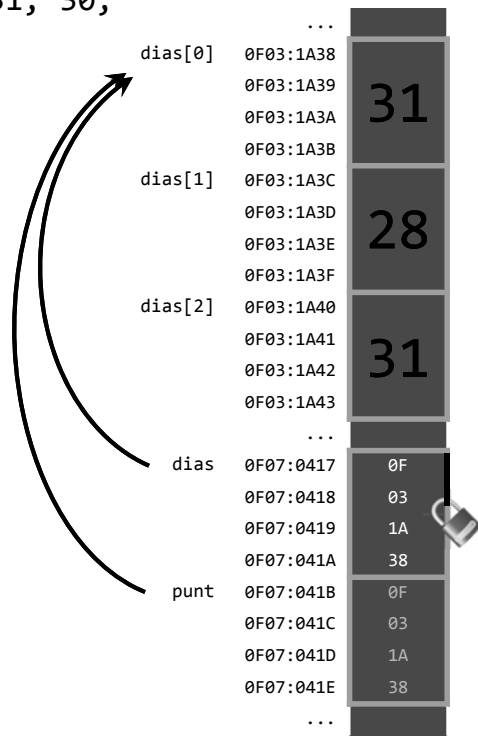
```
cout << *punt << endl; // Muestra 28 (segundo elemento)
```

A la dirección de memoria actual se le suman tantas unidades como bytes (4) ocupe en memoria un dato de ese tipo (int)



# Aritmética de punteros

```
int dias[12] = { 31, 28, 31, 30, 31, 30,
                31, 31, 30, 31, 30, 31 };
typedef int* tIntPtr;
tIntPtr punt = dias;
```

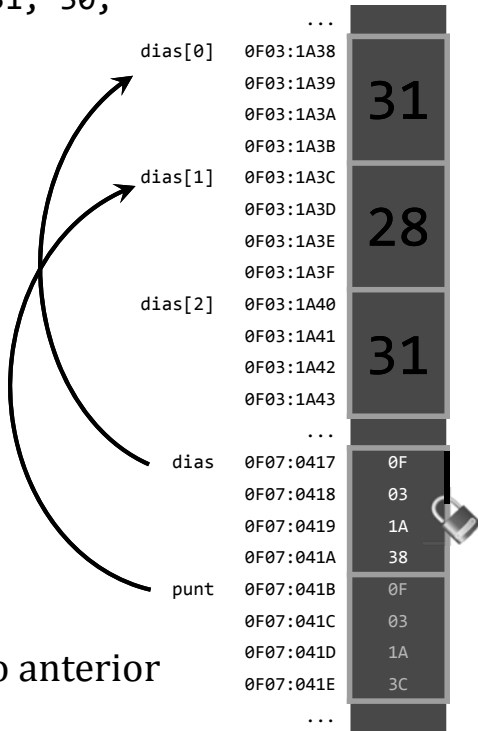


Luis Hernández Yáñez



# Aritmética de punteros

```
int dias[12] = { 31, 28, 31, 30, 31, 30,
                31, 31, 30, 31, 30, 31 };
typedef int* tIntPtr;
tIntPtr punt = dias;
punt++;
```



Luis Hernández Yáñez

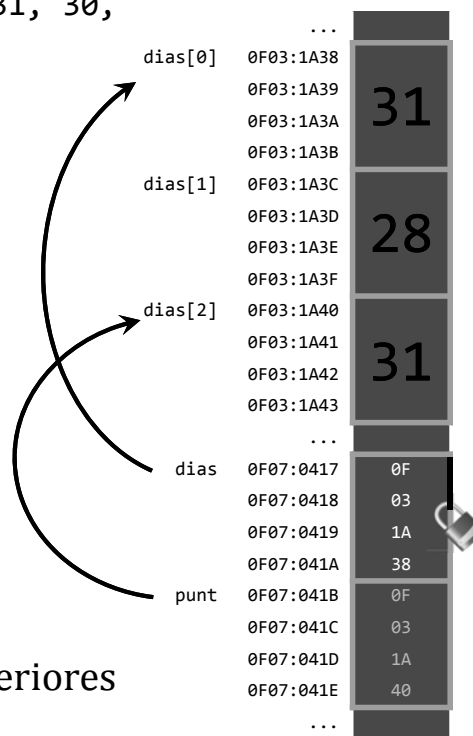


punt-- hace que apunte al elemento anterior



# Aritmética de punteros

```
int dias[12] = { 31, 28, 31, 30, 31, 30,  
                31, 31, 30, 31, 30, 31 };  
typedef int* tIntPtr;  
tIntPtr punt = dias;  
punt = punt + 2;
```



Restando pasamos a elementos anteriores

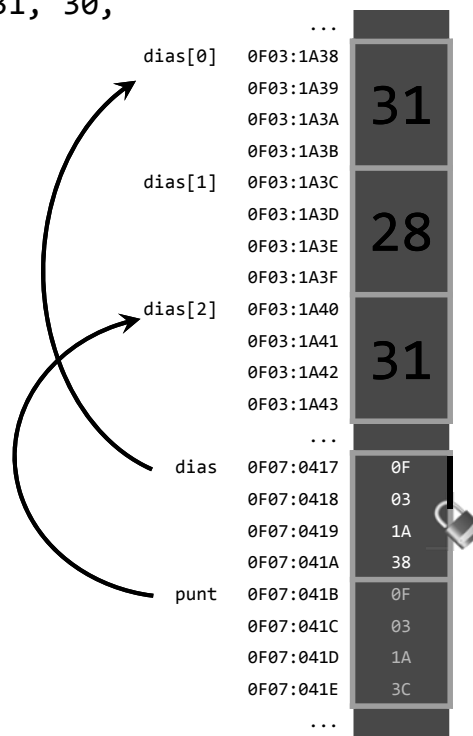
Luis Hernández Yáñez



# Aritmética de punteros

```
int dias[12] = { 31, 28, 31, 30, 31, 30,  
                31, 31, 30, 31, 30, 31 };  
typedef int* tIntPtr;  
tIntPtr punt = dias;  
punt = punt + 2;
```

```
int num = punt - dias;  
Nº de elementos entre los punteros
```



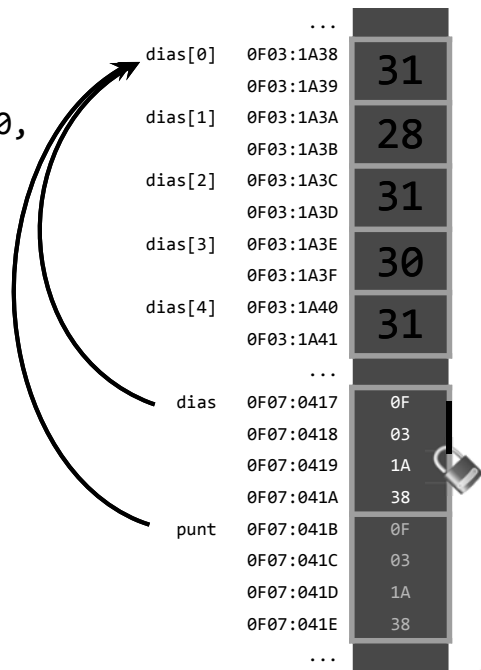
Luis Hernández Yáñez



# Aritmética de punteros

## Otro tipo base

```
short int (2 bytes)
short int dias[12] = {31, 28, 31, 30,
    31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
tSIPtr punt = dias;
```

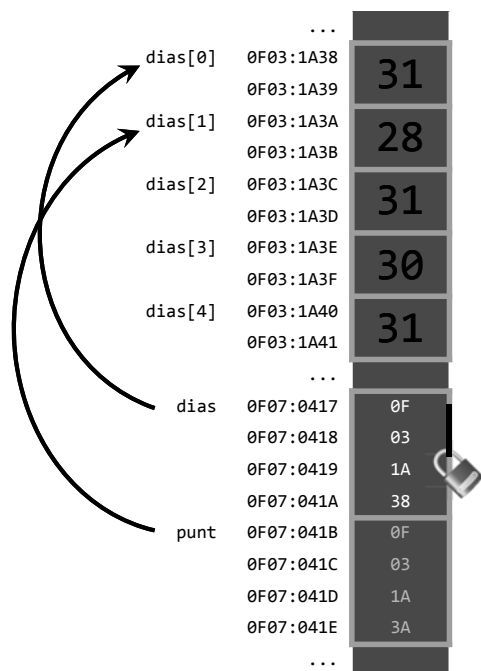


Luis Hernández Yáñez



# Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,
    31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
tSIPtr punt = dias;
punt++;
```

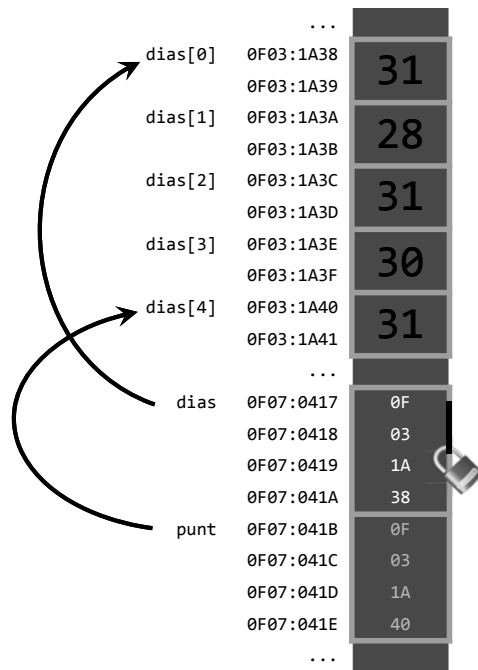


Luis Hernández Yáñez



# Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,  
    31, 30, 31, 31, 30, 31, 30, 31};  
typedef short int* tSIPtr;  
tSIPtr punt = dias;  
punt++;  
punt = punt + 3;
```

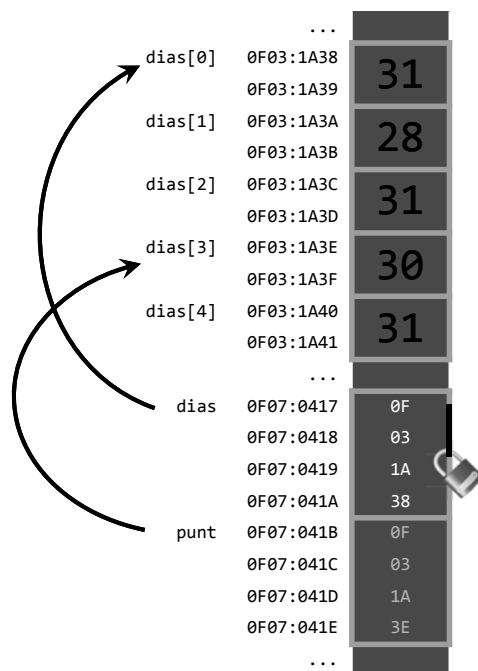


Luis Hernández Yáñez



# Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,  
    31, 30, 31, 31, 30, 31, 30, 31};  
typedef short int* tSIPtr;  
tSIPtr punt = dias;  
punt++;  
punt = punt + 3;  
punt--;
```

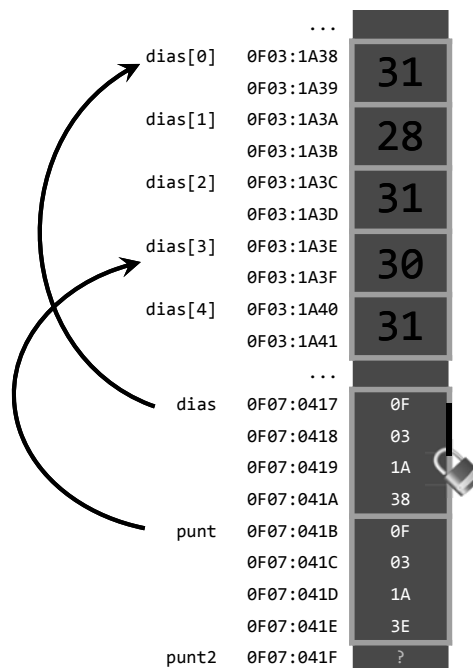


Luis Hernández Yáñez



# Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,
    31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
tSIPtr punt = dias;
punt++;
punt = punt + 3;
punt--;
tSIPtr punt2;
```

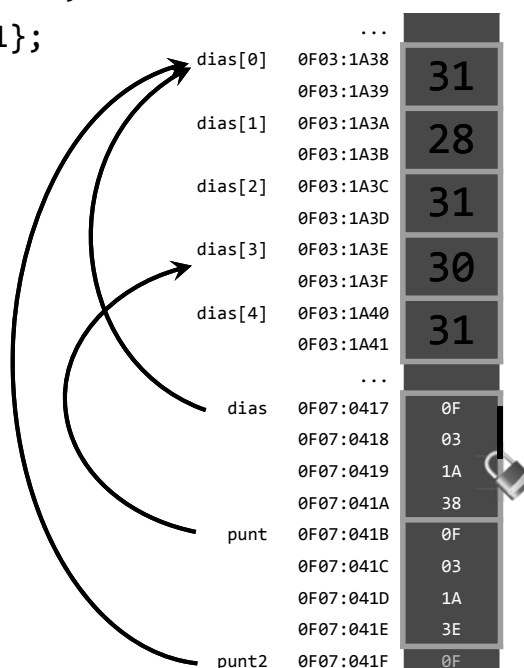


Luis Hernández Yáñez



# Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,
    31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
siPtr punt = dias;
punt++;
punt = punt + 3;
punt--;
tSIPtr punt2;
punt2 = dias;
```

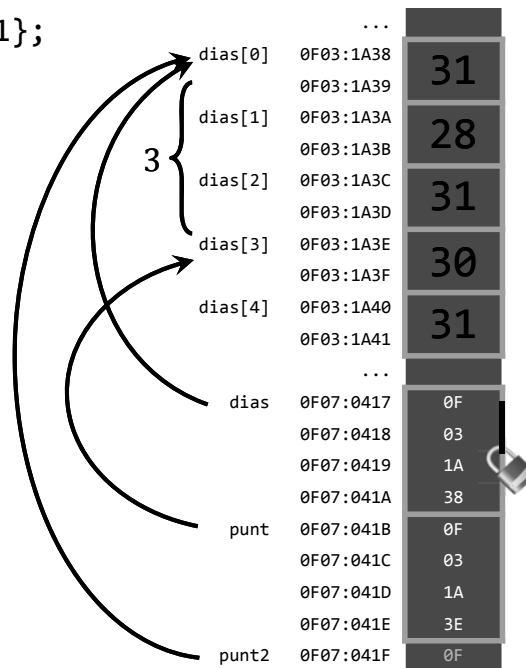


Luis Hernández Yáñez



# Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,  
    31, 30, 31, 31, 30, 31, 30, 31};  
typedef short int* tSIPtr;  
siPtr punt = dias;  
punt++;  
punt = punt + 3;  
punt--;  
tSIPtr punt2;  
punt2 = dias;  
cout << punt - punt2; // 3
```



Luis Hernández Yáñez



# Fundamentos de la programación

## Recorrido de arrays con punteros

Luis Hernández Yáñez





## Punteros como iteradores para arrays

```
const int MAX = 100;
typedef int tArray[MAX];
typedef struct {
    tArray elementos;
    int cont;
} tLista;
typedef int* tIntPtr;
tLista lista;
```

Usamos un puntero como *iterador* para recorrer el array:

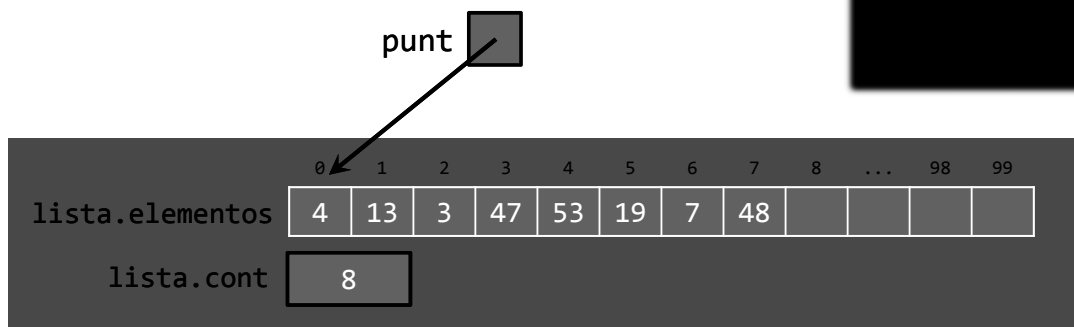
```
tIntPtr punt = lista.elementos;
for (int i = 0; i < lista.cont; i++) {
    cout << *punt << endl;
    punt++;
}
```

Luis Hernández Yáñez



# Punteros como iteradores para arrays

```
...
IntPtr punt = lista.elementos;
```



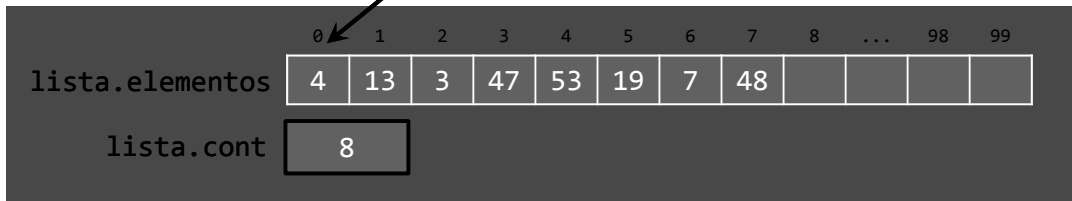
Luis Hernández Yáñez



# Punteros como iteradores para arrays

```
...  
for (int i = 0; i < lista.cont; i++) {  
    cout << *punt << endl;  
    punt++;  
}
```

i 0      punt  



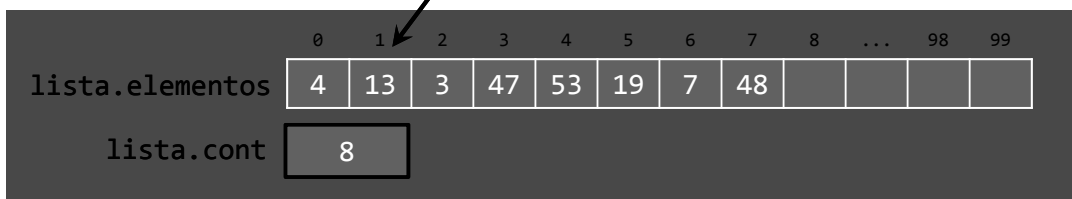
Luis Hernández Yáñez



# Punteros como iteradores para arrays

```
...  
for (int i = 0; i < lista.cont; i++) {  
    cout << *punt << endl;  
    punt++;  
}
```

i 1      punt  



Luis Hernández Yáñez

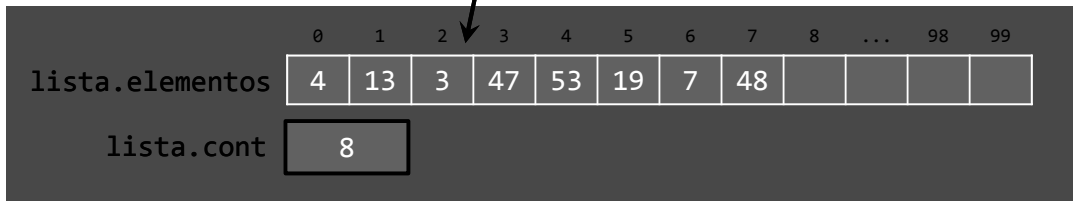


# Punteros como iteradores para arrays

```
...  
for (int i = 0; i < lista.cont; i++) {  
    cout << *punt << endl;  
    punt++;  
}
```

```
4  
13
```

i 2      punt  



Luis Hernández Yáñez

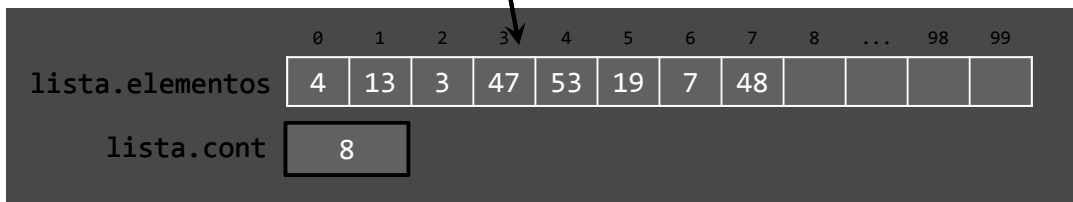


# Punteros como iteradores para arrays

```
...  
for (int i = 0; i < lista.cont; i++) {  
    cout << *punt << endl;  
    punt++;  
}
```

```
4  
13  
3
```

i 3      punt  



Luis Hernández Yáñez

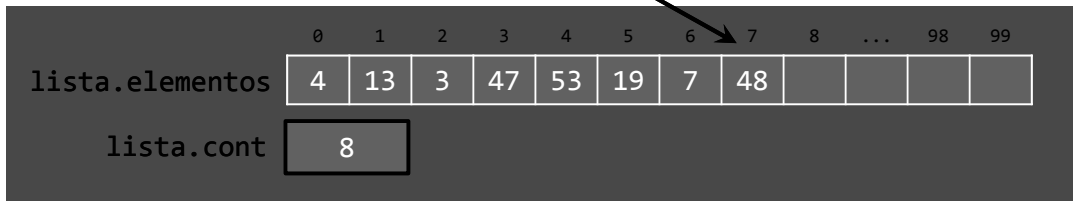


# Punteros como iteradores para arrays

```
...  
for (int i = 0; i < lista.cont; i++) {  
    cout << *punt << endl;  
    punt++;  
}
```

```
4  
13  
3  
47  
53  
19  
7
```

i  punt



Luis Hernández Yáñez

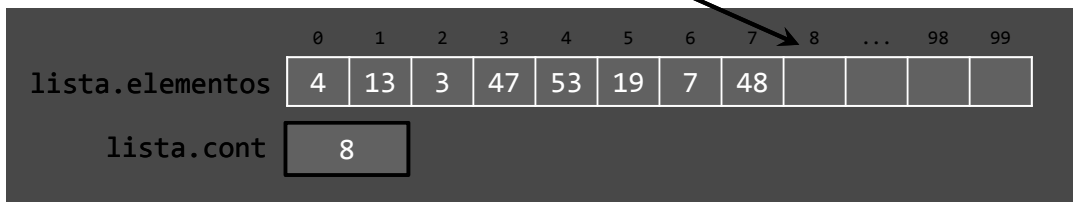


# Punteros como iteradores para arrays

```
...  
for (int i = 0; i < lista.cont; i++) {  
    cout << *punt << endl;  
    punt++;  
}
```

```
4  
13  
3  
47  
53  
19  
7  
48
```

i  punt



Luis Hernández Yáñez



## Referencias



## Referencias

---

### *Nombres alternativos para los datos*

Una referencia es una nueva forma de llamar a una variable

Nos permiten referirnos a una variable con otro identificador:

```
int x = 10;
```

```
int &z = x;
```

x y z son ahora la misma variable (comparten memoria)

Cualquier cambio en x afecta a z y cualquier cambio en z afecta a x

```
z = 30;
```

```
cout << x;
```

Las referencias se usan en el paso de parámetros por referencia



## Listas enlazadas



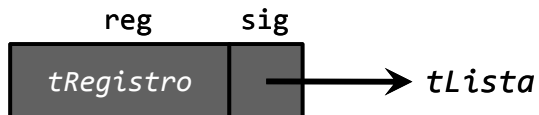
## Listas enlazadas

---

### *Una implementación dinámica de listas enlazadas*

Cada elemento de la lista apunta al siguiente elemento:

```
struct tNodo; // Declaración anticipada
typedef tNodo *tLista;
struct tNodo {
    tRegistro reg;
    tLista sig;
};
```



Una lista (`tLista`) es un puntero a un nodo

Si el puntero vale NULL, no apunta a ningún nodo: lista vacía

Un nodo (`tNodo`) es un elemento seguido de una lista

$Lista \begin{cases} Vacía \\ Elemento seguido de una lista \end{cases}$  ¡Definición recursiva!



# Implementación dinámica de listas enlazadas

Cada elemento de la lista en su nodo

Apuntará al siguiente elemento o a ninguno (NULL)

```
struct tNodo; // Declaración anticipada
typedef tNodo *tLista;
struct tNodo {
    tRegistro reg;
    tLista sig;
};
```

Además, un puntero al primer elemento (nodo) de la lista

```
tLista lista = NULL; // Lista vacía
```

lista 



# Implementación dinámica de listas enlazadas

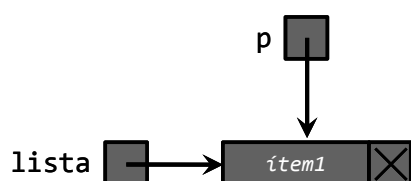
```
struct tNodo;
typedef tNodo *tLista;
struct tNodo {
    tRegistro reg;
    tLista sig;
};
tLista lista = NULL; // Lista vacía
lista = new tNodo;
lista->reg = nuevo();
lista->sig = NULL;
```

lista 



# Implementación dinámica de listas enlazadas

```
tLista lista = NULL; // Lista vacía  
lista = new tNodo;  
lista->reg = nuevo();  
lista->sig = NULL;  
tLista p;  
p = lista;
```

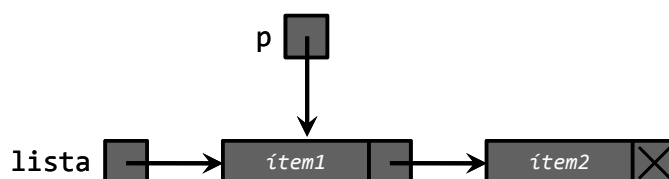


Luis Hernández Yáñez



# Implementación dinámica de listas enlazadas

```
tLista lista = NULL; // Lista vacía  
lista = new tNodo;  
lista->reg = nuevo();  
lista->sig = NULL;  
tLista p;  
p = lista;  
p->sig = new tNodo;  
p->sig->reg = nuevo();  
p->sig->sig = NULL;
```



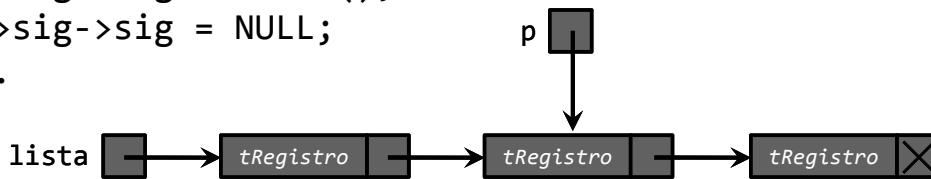
Luis Hernández Yáñez





# Implementación dinámica de listas enlazadas

```
tLista lista = NULL; // Lista vacía
lista = new tNodo;
lista->reg = nuevo();
lista->sig = NULL;
tLista p;
p = lista;
p->sig = new tNodo;
p->sig->reg = nuevo();
p->sig->sig = NULL;
p = p->sig;
p->sig = new tNodo;
p->sig->reg = nuevo();
p->sig->sig = NULL;
...
```



Luis Hernández Yáñez



# Implementación dinámica de listas enlazadas

Usamos la memoria que necesitamos, ni más ni menos



Tantos elementos, tantos nodos hay en la lista

*¡Pero perdemos el acceso directo!*

Algunas operaciones de la lista se complican y otras no

A continuación tienes el módulo de lista implementado como lista enlazada...

Luis Hernández Yáñez



# Ejemplo de lista enlazada

listaenlazada.h

```
struct tNodo;
typedef tNodo *tLista;
struct tNodo {
    tRegistro reg;
    tLista sig;
};

const string BD = "bd.txt";

void mostrar(tLista lista);
void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int code, bool &ok);
tLista buscar(tLista lista, int code); // Devuelve puntero
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
void destruir(tLista &lista); // Liberar la memoria dinámica
```

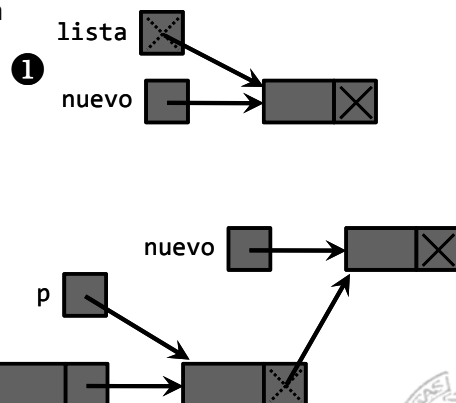
Luis Hernández Yáñez



# Ejemplo de lista enlazada

listaenlazada.cpp

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    tLista nuevo = new tNodo;
    if (nuevo == NULL) {
        ok = false; // No hay más memoria dinámica
    }
    else {
        nuevo->reg = registro;
        nuevo->sig = NULL;
        if (lista == NULL) { // Lista vacía
            ① lista = nuevo;
        }
        else {
            tLista p = lista;
            // Localizamos el último nodo...
            ② while (p->sig != NULL) {
                p = p->sig;
            }
            p->sig = nuevo;
        }
    }
} ...
```

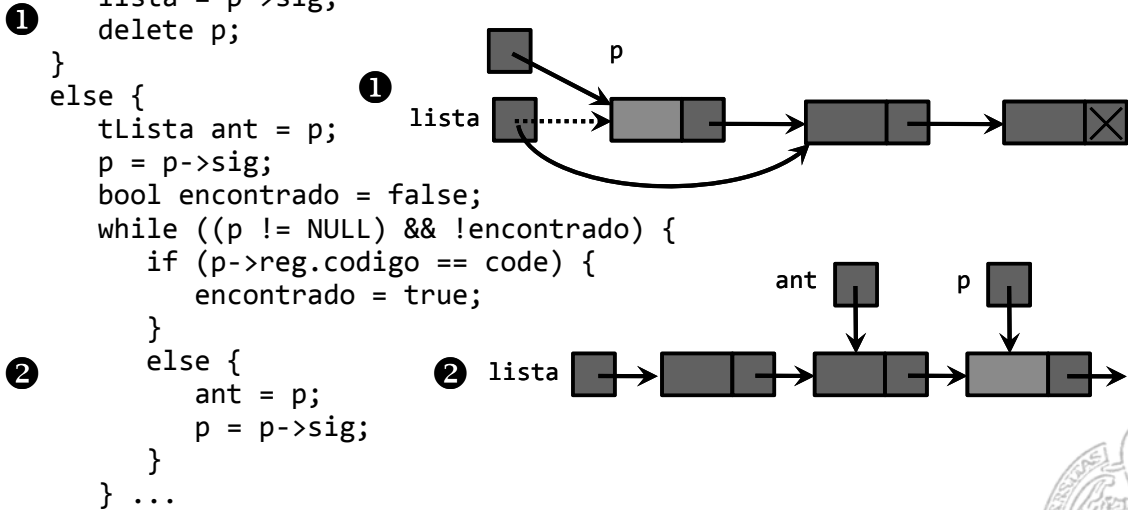


Luis Hernández Yáñez



# Ejemplo de lista enlazada

```
void eliminar(tLista &lista, int code, bool &ok) {
    ok = true;
    tLista p = lista;
    if (p == NULL) {
        ok = false; // Lista vacía
    }
    else if (p->reg.codigo == code) { // El primero
        lista = p->sig;
        delete p;
    }
    else {
        tLista ant = p;
        p = p->sig;
        bool encontrado = false;
        while ((p != NULL) && !encontrado) {
            if (p->reg.codigo == code) {
                encontrado = true;
            }
            else {
                ant = p;
                p = p->sig;
            }
        }
        ...
    }
}
```

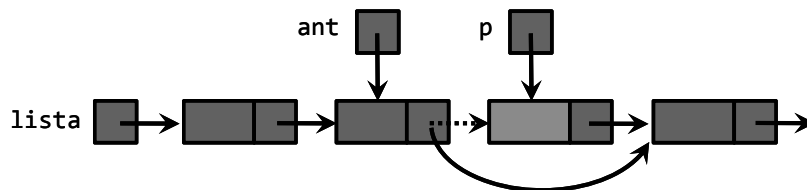


Luis Hernández Yáñez



# Ejemplo de lista enlazada

```
if (!encontrado) {
    ok = false; // No existe ese código
}
else {
    ant->sig = p->sig;
    delete p;
}
}
...
}
```



Luis Hernández Yáñez



# Ejemplo de lista enlazada

---

```
tLista buscar(tLista lista, int code) {
// Devuelve un puntero al nodo, o NULL si no se encuentra
    tLista p = lista;
    bool encontrado = false;
    while ((p != NULL) && !encontrado) {
        if (p->reg.codigo == code) {
            encontrado = true;
        }
        else {
            p = p->sig;
        }
    }
    return p;
}

void mostrar(tLista lista) {
    cout << endl << "Elementos de la lista:" << endl
        << "-----" << endl;
    tLista p = lista;
    while (p != NULL) {
        mostrar(p->reg);
        p = p->sig;
    }
} ...
```

Luis Hernández Yáñez



# Ejemplo de lista enlazada

---

```
void cargar(tLista &lista, bool &ok) {
    ifstream archivo;
    char aux;
    ok = true;
    lista = NULL;
    archivo.open(BD.c_str());
    if (!archivo.is_open()) {
        ok = false;
    }
    else {
        tRegistro registro;
        tLista ult = NULL;
        archivo >> registro.codigo;
        while (registro.codigo != -1) {
            archivo >> registro.valor;
            archivo.get(aux); // Saltamos el espacio
            getline(archivo, registro.nombre);
            ...
        }
    }
}
```

Luis Hernández Yáñez



# Ejemplo de lista enlazada

---

```
    if (lista == NULL) {
        lista = new tNodo;
        ult = lista;
    }
    else {
        ult->sig = new tNodo;
        ult = ult->sig;
    }
    ult->reg = registro;
    ult->sig = NULL;
    archivo >> registro.codigo;
}
archivo.close();
}
return ok;
} ...
```



# Ejemplo de lista enlazada

---

```
void guardar(tLista lista) {
    ofstream archivo;
    archivo.open(BD);
    tLista p = lista;
    while (p != NULL) {
        archivo << p->registro.codigo << " ";
        archivo << p->registro.valor << " ";
        archivo << p->registro.nombre << endl;
        p = p->sig;
    }
    archivo.close();
}

void destruir(tLista &lista) {
    tLista p;
    while (lista != NULL) {
        p = lista;
        lista = lista->sig;
        delete p;
    }
}
```








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.



# 10

## Introducción a la recursión

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Concepto de recursión	983
Algoritmos recursivos	986
Funciones recursivas	987
Diseño de funciones recursivas	989
Modelo de ejecución	990
La pila del sistema	992
La pila y las llamadas a función	994
Ejecución de la función factorial()	1005
Tipos de recursión	1018
Recursión simple	1019
Recursión múltiple	1020
Recursión anidada	1022
Recursión cruzada	1026
Código del subprograma recursivo	1027
Parámetros y recursión	1032
Ejemplos de algoritmos recursivos	1034
Búsqueda binaria	1035
Torres de Hanoi	1038
Recursión frente a iteración	1043
Estructuras de datos recursivas	1045



## Recursión

Luis Hernández Yáñez

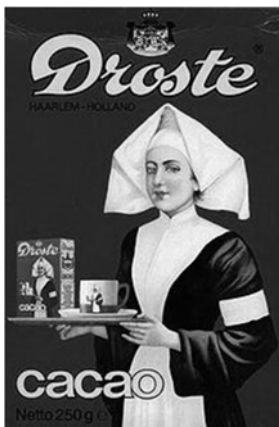


## Concepto de recursión

### *Recursión (recursividad, recurrencia)*

*Definición recursiva: En la definición aparece lo que se define*

$$\text{Factorial}(N) = N \times \text{Factorial}(N-1) \quad (N \geq 0)$$



(wikipedia.org)

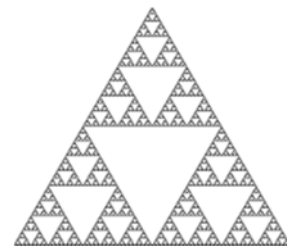


La cámara graba lo que graba

([http://farm1.static.flickr.com/83/229219543\\_edf740535b.jpg](http://farm1.static.flickr.com/83/229219543_edf740535b.jpg))

La imagen del paquete aparece dentro del propio paquete,... ¡hasta el infinito!

Cada triángulo está formado por otros triángulos más pequeños



(wikipedia.org)



Las matrioskas rusas

Luis Hernández Yáñez





# Definiciones recursivas

---

$$Factorial(N) = N \times Factorial(N-1)$$

El factorial se define en función de sí mismo

Los programas no pueden manejar la recursión infinita

La definición recursiva debe adjuntar uno o más casos base

*Caso base*: aquel en el que no se utiliza la definición recursiva

Proporcionan puntos finales de cálculo:

$$Factorial(N) \begin{cases} N \times Factorial(N-1) & \text{si } N > 0 & \text{Caso recursivo (inducción)} \\ 1 & \text{si } N = 0 & \text{Caso base (o de parada)} \end{cases}$$

El valor de N se va aproximando al valor del caso base (0)



# Fundamentos de la programación

---

## Algoritmos recursivos



# Algoritmos recursivos

## Funciones recursivas

Una función puede implementar un algoritmo recursivo

La función se llamará a sí misma si no se ha llegado al caso base

$$\text{Factorial}(N) \begin{cases} 1 & \text{si } N = 0 \\ N \times \text{Factorial}(N-1) & \text{si } N > 0 \end{cases}$$

```
long long int factorial(int n) {
    long long int resultado;
    if (n == 0) { // Caso base
        resultado = 1;
    }
    else {
        resultado = n * factorial(n - 1);
    }
    return resultado;
}
```

Luis Hernández Yáñez



# Algoritmos recursivos

factorial.cpp

## Funciones recursivas

```
long long int factorial(int n) {
    long long int resultado;
    if (n == 0) { // Caso base
        resultado = 1;
    }
    else {
        resultado = n * factorial(n - 1);
    }
    return resultado;
}
```

factorial(5) → 5 x factorial(4) → 5 x 4 x factorial(3)  
→ 5 x 4 x 3 x factorial(2) → 5 x 4 x 3 x 2 x factorial(1)  
→ 5 x 4 x 3 x 2 x 1 x factorial(0) → 5 x 4 x 3 x 2 x 1 x 1  
→ 120 Caso base

```
1
1
2
6
24
120
720
5040
40320
362880
3628800
39916800
479001600
6227020800
87178291200
1307674368000
20922789888000
355687428096000
6402373705728000
121645100408832000
```

Luis Hernández Yáñez



# Algoritmos recursivos

---

## *Diseño de funciones recursivas*

Una función recursiva debe satisfacer tres condiciones:

- ✓ Caso(s) base: Debe haber al menos un caso base de parada
- ✓ Inducción: Paso recursivo que provoca una llamada recursiva  
Debe ser correcto para distintos parámetros de entrada
- ✓ Convergencia: Cada paso recursivo debe acercar a un caso base

Se describe el problema en términos de problemas *más sencillos*

$$\text{Factorial}(N) \begin{cases} 1 & \text{si } N = 0 \\ N \times \text{Factorial}(N-1) & \text{si } N > 0 \end{cases}$$

Función `factorial()`: tiene caso base ( $N = 0$ ), siendo correcta para  $N$  es correcta para  $N+1$  (*inducción*) y se acerca cada vez más al caso base ( $N-1$  está más cerca de 0 que  $N$ )



# Fundamentos de la programación

---

## Modelo de ejecución



# Modelo de ejecución

```
long long int factorial(int n) {  
    long long int resultado;  
    if (n == 0) { // Caso base  
        resultado = 1;  
    }  
    else {  
        resultado = n * factorial(n - 1);  
    }  
    return resultado;  
}
```

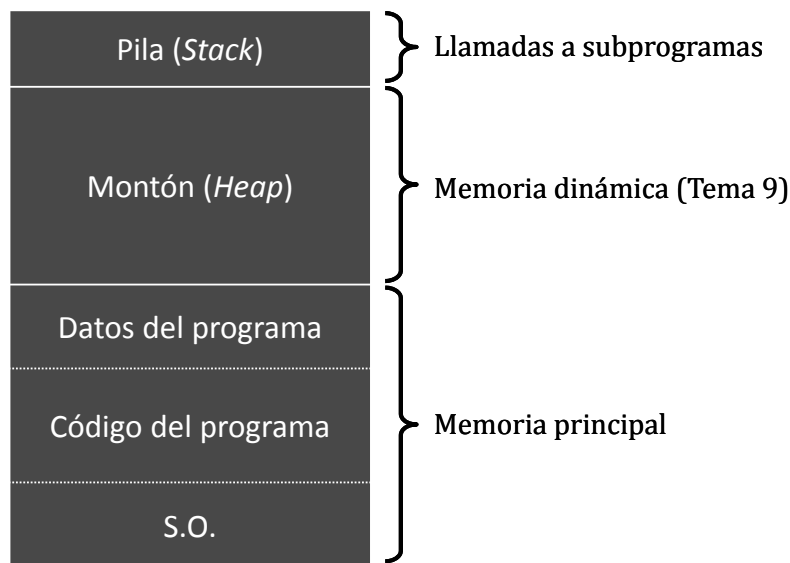
Cada llamada recursiva fuerza una nueva ejecución de la función  
Cada llamada utiliza sus propios parámetros por valor y variables locales (n y resultado en este caso)  
En las llamadas a la función se utiliza la pila del sistema para mantener los datos locales y la dirección de vuelta

Luis Hernández Yáñez



# La pila del sistema (*stack*)

Regiones de memoria que distingue el sistema operativo:



Luis Hernández Yáñez

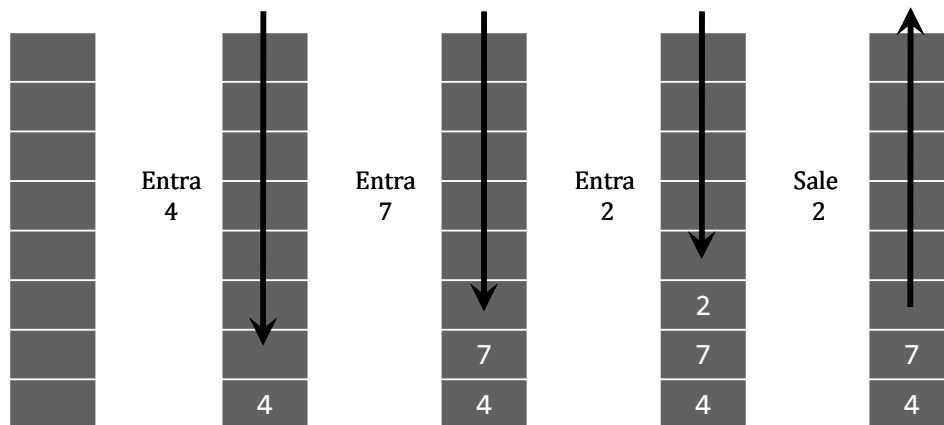


# La pila del sistema (*stack*)

Mantiene los datos locales de la función y la dirección de vuelta

Estructura de tipo *pila*: lista LIFO (*last-in first-out*)

El último que entra es el primero que sale:



Luis Hernández Yáñez



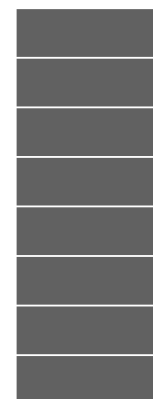
# La pila y las llamadas a función

Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
    ...
    b = funcB(a);
    ...
    return b;
}
int main() {
    ...
    cout << funcA(4);
    ...
}
```

<DIR1>

Llamada a función:  
Entra la dirección de vuelta



Pila

Luis Hernández Yáñez

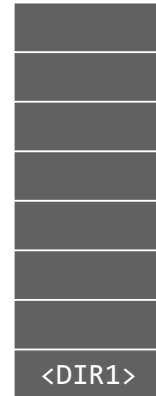


# La pila y las llamadas a función

## Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
<DIR1>   cout << funcA(4);
    ...
}
```

← Entrada en la función:  
Se alojan los datos locales



Pila



# La pila y las llamadas a función

## Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
<DIR1>   cout << funcA(4);
    ...
}
```

← Llamada a función:  
Entra la dirección de vuelta



Pila



# La pila y las llamadas a función

## Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
    ...
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
    ...
<DIR1>   cout << funcA(4);
    ...
}
```



Entrada en la función:  
Se alojan los datos locales



Pila



# La pila y las llamadas a función

## Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
    ...
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
    ...
<DIR1>   cout << funcA(4);
    ...
}
```



Vuelta de la función:  
Se eliminan los datos locales



Pila



# La pila y las llamadas a función

## Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
    ...
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
    ...
<DIR1>   cout << funcA(4);
    ...
}
```

← Vuelta de la función:  
Sale la dirección de vuelta



Pila

Luis Hernández Yáñez



# La pila y las llamadas a función

## Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
    ...
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
    ...
<DIR1>   cout << funcA(4);
    ...
}
```

← La ejecución continúa  
en esa dirección



Pila

Luis Hernández Yáñez





# La pila y las llamadas a función

## Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
    ...
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
    ...
<DIR1>   cout << funcA(4);
    ...
}
```

← Vuelta de la función:  
Se eliminan los datos locales



Pila

Luis Hernández Yáñez

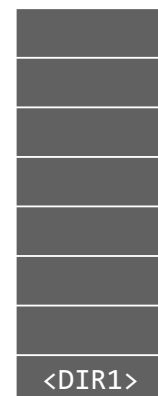


# La pila y las llamadas a función

## Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
    ...
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
    ...
<DIR1>   cout << funcA(4);
    ...
}
```

← Vuelta de la función:  
Sale la dirección de vuelta



Pila

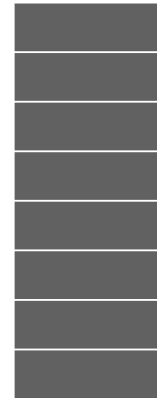
Luis Hernández Yáñez



# La pila y las llamadas a función

Datos locales y direcciones de vuelta

```
...
int funcB(int x) {
    ...
    return x;
}
int funcA(int a) {
    int b;
    ...
<DIR2>   b = funcB(a);
    ...
    return b;
}
int main() {
    ...
<DIR1>   cout << funcA(4); ← La ejecución continúa
    ...                                     en esa dirección
```

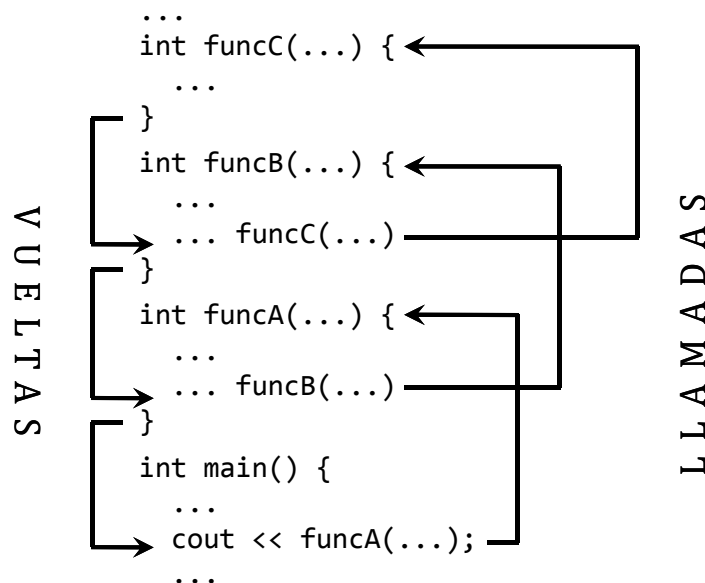


Pila



# La pila y las llamadas a función

Mecanismo de pila adecuado para llamadas a funciones anidadas:  
Las llamadas terminan en el orden contrario a como se llaman



Pila



# Ejecución de la función factorial()

```
long long int factorial(int n) {  
    long long int resultado;  
    if (n == 0) { // Caso base  
        resultado = 1;  
    }  
    else {  
        resultado = n * factorial(n - 1);  
    }  
    return resultado;  
}
```

```
cout << factorial(5) << endl;
```

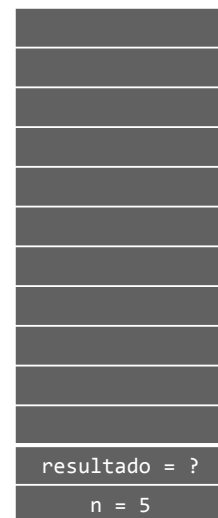


Obviaremos las direcciones de vuelta en la pila



# Ejecución de la función factorial()

```
factorial(5)
```

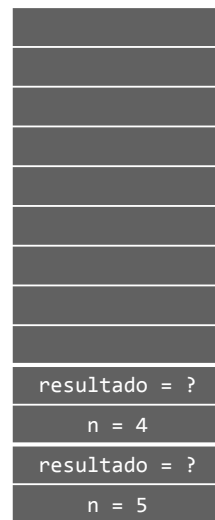


Pila



# Ejecución de la función factorial()

```
factorial(5)
  ↳ factorial(4)
```

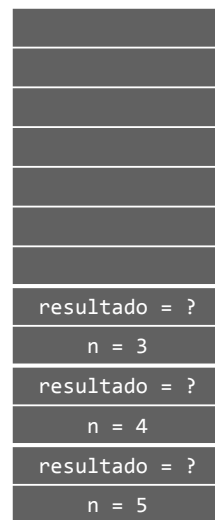


Pila



# Ejecución de la función factorial()

```
factorial(5)
  ↳ factorial(4)
    ↳ factorial(3)
```



Pila



# Ejecución de la función factorial()

```
factorial(5)
  ↳ factorial(4)
    ↳ factorial(3)
      ↳ factorial(2)
```

resultado = ?
n = 2
resultado = ?
n = 3
resultado = ?
n = 4
resultado = ?
n = 5

Pila



# Ejecución de la función factorial()

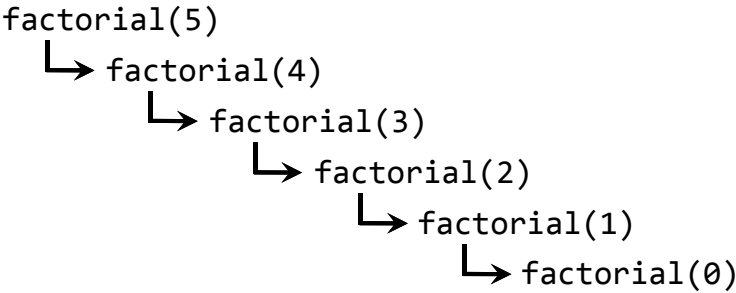
```
factorial(5)
  ↳ factorial(4)
    ↳ factorial(3)
      ↳ factorial(2)
        ↳ factorial(1)
```

resultado = ?
n = 1
resultado = ?
n = 2
resultado = ?
n = 3
resultado = ?
n = 4
resultado = ?
n = 5

Pila



# Ejecución de la función factorial()



resultado = 1
n = 0
resultado = ?
n = 1
resultado = ?
n = 2
resultado = ?
n = 3
resultado = ?
n = 4
resultado = ?
n = 5

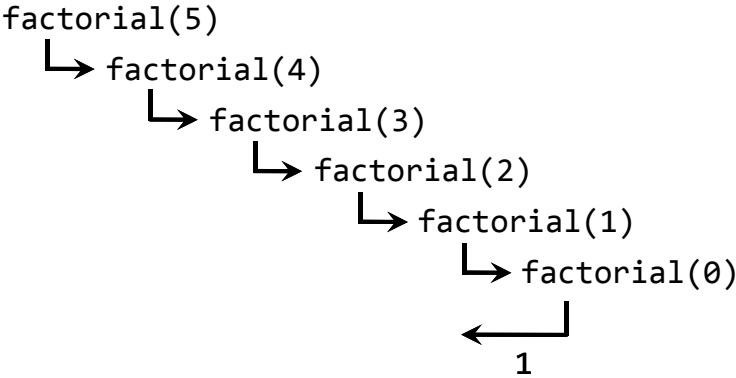
Pila



Luis Hernández Yáñez



# Ejecución de la función factorial()



resultado = 1
n = 1
resultado = ?
n = 2
resultado = ?
n = 3
resultado = ?
n = 4
resultado = ?
n = 5

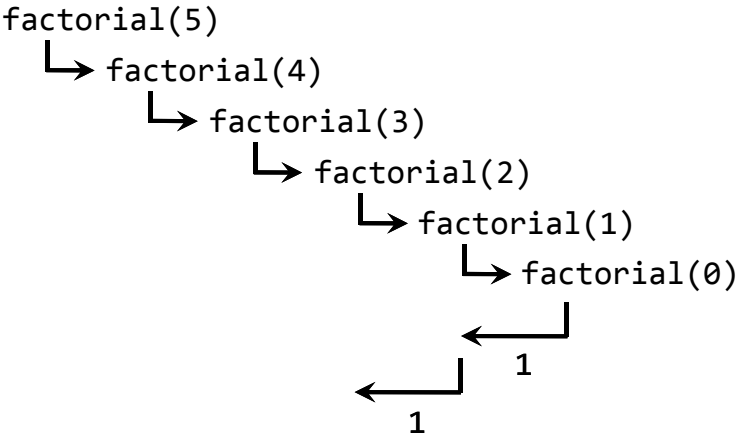
Pila



Luis Hernández Yáñez



# Ejecución de la función factorial()



resultado = 2
n = 2
resultado = ?
n = 3
resultado = ?
n = 4
resultado = ?
n = 5

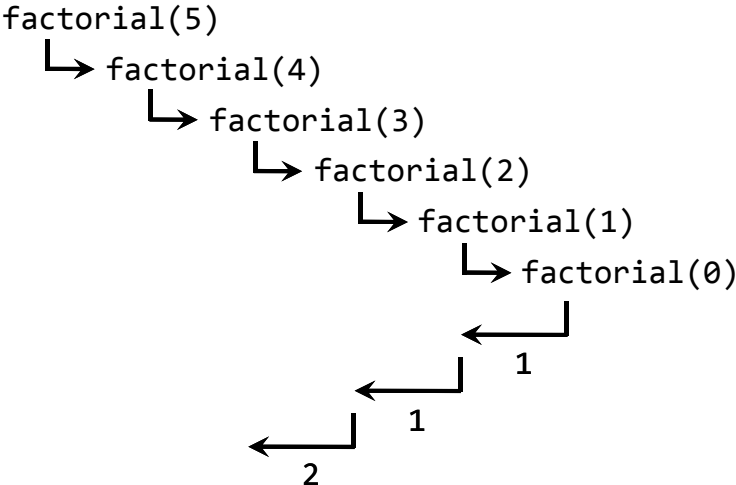
Pila



Luis Hernández Yáñez



# Ejecución de la función factorial()



resultado = 6
n = 3
resultado = ?
n = 4
resultado = ?
n = 5

Pila



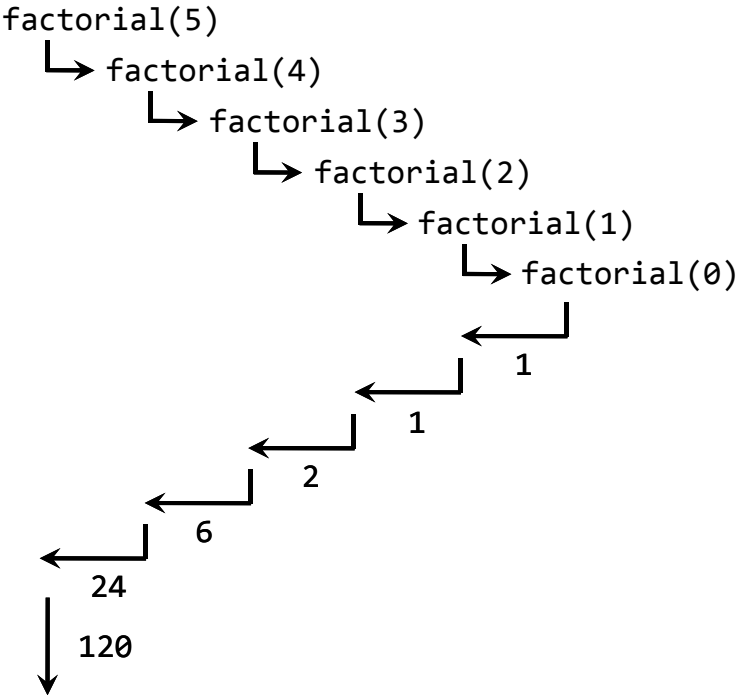
Luis Hernández Yáñez







# Ejecución de la función factorial()



Pila

Luis Hernández Yáñez



# Fundamentos de la programación

## Tipos de recursión

Luis Hernández Yáñez



# Recursión simple

---

Sólo hay una llamada recursiva

Ejemplo: Cálculo del factorial de un número entero positivo

```
long long int factorial(int n) {  
    long long int resultado;  
    if (n == 0) { // Caso base  
        resultado = 1;  
    }  
    else {  
        resultado = n * factorial(n - 1);  
    }  
    return resultado;  
}
```

Una sola llamada recursiva



# Recursión múltiple

---

Varias llamadas recursivas

Ejemplo: Cálculo de los números de *Fibonacci*

$$\text{Fib}(n) \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{si } n > 1 \end{cases}$$

Dos llamadas recursivas



# Recursión múltiple

fibonacci.cpp

```
...
int main() {
    for (int i = 0; i < 20; i++) {
        cout << fibonacci(i) << endl;
    }
    return 0;
}
```

$$\text{Fib}(n) \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{si } n > 1 \end{cases}$$

```
int fibonacci(int n) {
    int resultado;
    if (n == 0) {
        resultado = 0;
    }
    else if (n == 1) {
        resultado = 1;
    }
    else {
        resultado = fibonacci(n - 1) + fibonacci(n - 2);
    }
    return resultado;
}
```

0  
1  
1  
2  
3  
5  
8  
13  
21  
34  
55  
89  
144  
233  
377  
610  
987  
1597  
2584  
4181

Luis Hernández Yáñez



# Recursión anidada

En una llamada recursiva alguno de los argumentos es otra llamada  
Ejemplo: Cálculo de los números de Ackermann:

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

Argumento que es una llamada recursiva

Luis Hernández Yáñez




# Recursión anidada

## Números de Ackermann

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

```
...
int ackermann(int m, int n) {
    int resultado;
    if (m == 0) {
        resultado = n + 1;
    }
    else if (n == 0) {
        resultado = ackermann(m - 1, 1);
    }
    else {
        resultado = ackermann(m - 1, ackermann(m, n - 1));
    }
    return resultado;
}
```

 Pruébalo con números muy bajos:  
Se generan MUCHAS llamadas recursivas



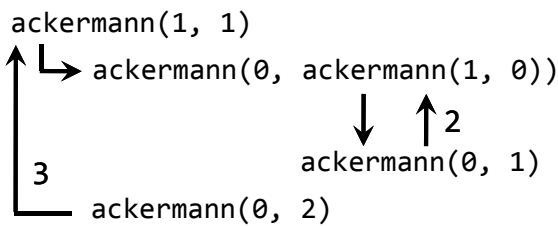
Luis Hernández Yáñez



# Recursión anidada

## Números de Ackermann

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



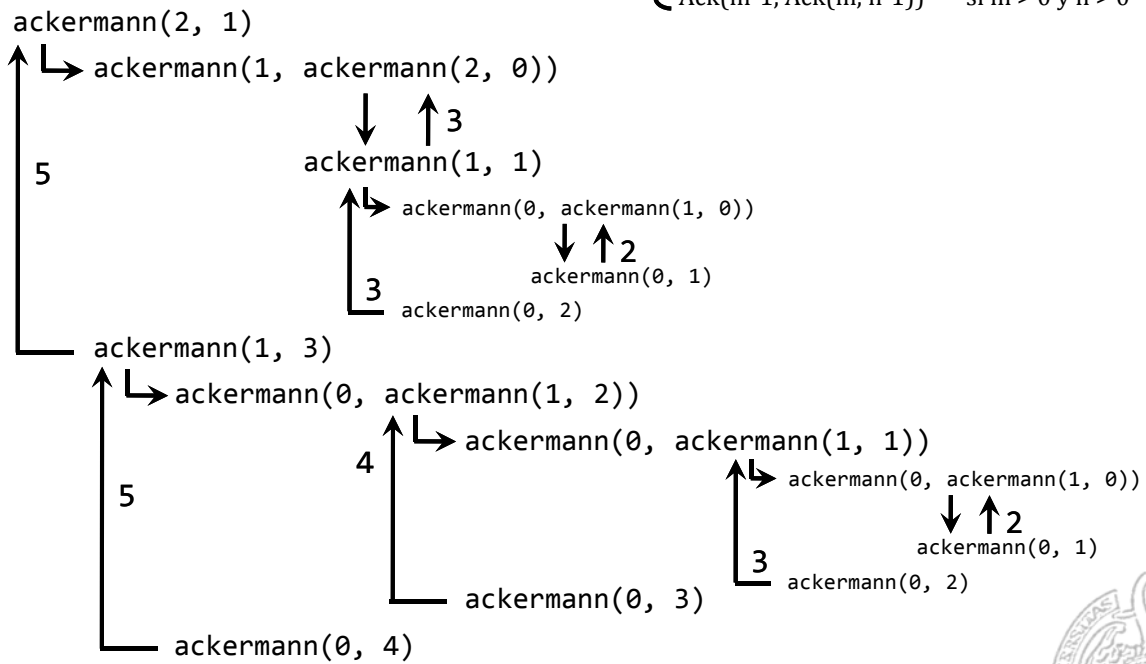
Luis Hernández Yáñez



# Recursión anidada

## Números de Ackermann

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



Luis Hernández Yáñez



# Fundamentos de la programación

## Código del subprograma recursivo

Luis Hernández Yáñez



# Código del subprograma recursivo

## *Código anterior y posterior a la llamada recursiva*

```
{  
  Código anterior  
  Llamada recursiva  
  Código posterior  
}
```

### *Código anterior*

Se ejecuta para las distintas entradas antes que el *código posterior*

### *Código posterior*

Se ejecuta para las distintas entradas tras llegarse al caso base

El código anterior se ejecuta en orden directo para las distintas entradas, mientras que el código posterior lo hace en orden inverso

Si no hay código anterior: *recursión por delante*

Si no hay código posterior: *recursión por detrás*



# Código del subprograma recursivo

## *Código anterior y posterior a la llamada recursiva*

```
void func(int n) {  
  if (n > 0) { // Caso base: n == 0  
    cout << "Entrando (" << n << ")" << endl; // Código anterior  
    func(n - 1); // Llamada recursiva  
    cout << "Saliendo (" << n << ")" << endl; // Código posterior  
  }  
}
```

→ `func(5);`

El código anterior se ejecuta para los sucesivos valores de n (5, 4, 3, ...)

El código posterior al revés (1, 2, 3, ...)

```
Entrando (5)  
Entrando (4)  
Entrando (3)  
Entrando (2)  
Entrando (1)  
Saliendo (1)  
Saliendo (2)  
Saliendo (3)  
Saliendo (4)  
Saliendo (5)
```



# Código del subprograma recursivo

directo.cpp

## Recorrido de los elementos de una lista (directo)

El código anterior a la llamada procesa la lista en su orden:

```
...
void mostrar(tLista lista, int pos);

int main() {
    tLista lista;
    lista.cont = 0;
    // Carga del array...
    mostrar(lista, 0);

    return 0;
}

void mostrar(tLista lista, int pos) {
    if (pos < lista.cont) {
        cout << lista.elementos[pos] << endl;
        mostrar(lista, pos + 1);
    }
}
```

1  
3  
8  
13  
17  
22  
23  
39  
52  
55

Luis Hernández Yáñez



# Código del subprograma recursivo

inverso.cpp

## Recorrido de los elementos de una lista (inverso)

El código posterior procesa la lista en el orden inverso:

```
...
void mostrar(tLista lista, int pos);

int main() {
    tLista lista;
    lista.cont = 0;
    // Carga del array...
    mostrar(lista, 0);

    return 0;
}

void mostrar(tLista lista, int pos) {
    if (pos < lista.cont) {
        mostrar(lista, pos + 1);
        cout << lista.elementos[pos] << endl;
    }
}
```

55  
52  
39  
23  
22  
17  
13  
8  
3  
1

Luis Hernández Yáñez



## Parámetros y recursión



## Parámetros y recursión

---

### *Parámetros por valor y por referencia*

Parámetros por valor: cada llamada usa los suyos propios

Parámetros por referencia: misma variable en todas las llamadas

Recogen resultados que transmiten entre las llamadas

```
void factorial(int n, int &fact) {
    if (n == 0) {
        fact = 1;
    }
    else {
        factorial(n - 1, fact);
        fact = n * fact;
    }
}
```

Cuando  $n$  es 0, el argumento de `fact` toma el valor 1

Al volver se le va multiplicando por los demás  $n$  (distintos)





## Ejemplos de algoritmos recursivos



## Búsqueda binaria

---

Parte el problema en subproblemas más pequeños

Aplica el mismo proceso a cada subproblema

Naturaleza recursiva (casos base: encontrado o no queda lista)

*Partimos de la lista completa*

*Si no queda lista... terminar (lista vacía: no encontrado)*

*En caso contrario...*

*Comprobar si el elemento en la mitad es el buscado*

*Si es el buscado... terminar (encontrado)*

*Si no...*

*Si el buscado es menor que el elemento mitad...*

*Repetir con la primera mitad de la lista*

*Si el buscado es mayor que el elemento mitad...*

*Repetir con la segunda mitad de la lista*

→ La repetición se consigue con las llamadas recursivas




# Búsqueda binaria

Dos índices que indiquen el inicio y el final de la sublista:

```
int buscar(tLista lista, int buscado, int ini, int fin)
// Devuelve el índice (0, 1, ...) o -1 si no está
```

¿Cuáles son los casos base?

- ✓ Que ya no quede sublista ( $ini > fin$ ) → No encontrado
- ✓ Que se encuentre el elemento

 Repasa en el Tema 7 cómo funciona y cómo se implementó iterativamente la búsqueda binaria (compárala con esta)



# Búsqueda binaria

binaria.cpp

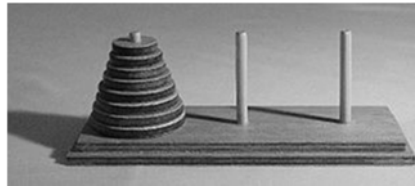
```
int buscar(tLista lista, int buscado, int ini, int fin) {
    int pos = -1;
    if (ini <= fin) {
        int mitad = (ini + fin) / 2;
        if (buscado == lista.elementos[mitad]) {
            pos = mitad;
        }
        else if (buscado < lista.elementos[mitad]) {
            pos = buscar(lista, buscado, ini, mitad - 1);
        }
        else {
            pos = buscar(lista, buscado, mitad + 1, fin);
        }
    }
    return pos;
}
```

Llamada: `pos = buscar(lista, valor, 0, lista.cont - 1);`



# Las Torres de Hanoi

*Cuenta una leyenda que en un templo de Hanoi se dispusieron tres pilares de diamante y en uno de ellos 64 discos de oro, de distintos tamaños y colocados por orden de tamaño con el mayor debajo*



Torre de ocho discos (wikipedia.org)

*Cada monje, en su turno, debía mover un único disco de un pilar a otro, para con el tiempo conseguir entre todos llevar la torre del pilar inicial a uno de los otros dos; respetando una única regla: nunca poner un disco sobre otro de menor tamaño*

*Cuando lo hayan conseguido, ¡se acabará el mundo!*

[Se requieren al menos  $2^{64}-1$  movimientos; si se hiciera uno por segundo, se terminaría en más de 500 mil millones de años]

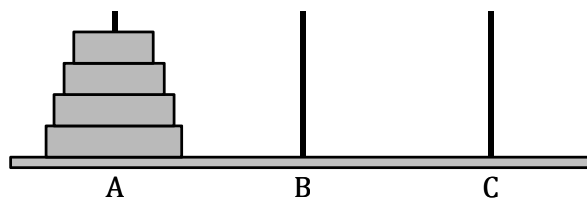


# Las Torres de Hanoi

Queremos resolver el *juego* en el menor número de pasos posible

¿Qué disco hay que mover en cada paso y a dónde?

Identifiquemos los elementos (torre de cuatro discos):



Cada pilar se identifica con una letra

*Mover del pilar X al pilar Y:*

Coger el disco superior de X y ponerlo encima de los que haya en Y



# Las Torres de Hanoi

Resolución del problema en base a problemas más pequeños

Mover N discos del pilar A al pilar C:

Mover N-1 discos del pilar A al pilar B

Mover el disco del pilar A al pilar C

Mover N-1 discos del pilar B al pilar C

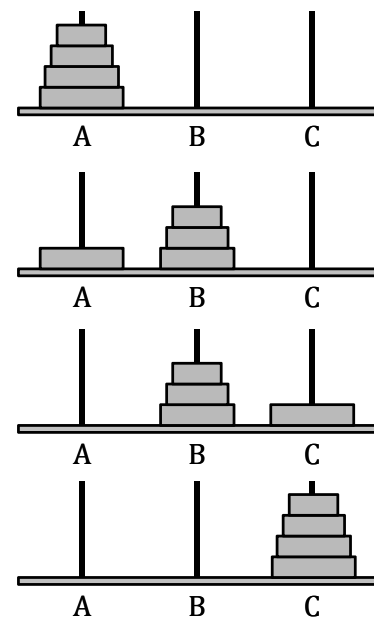
Para llevar N discos de un pilar *origen* a otro *destino* se usa el tercero como *auxiliar*

Mover N-1 discos del *origen* al *auxiliar*

Mover el disco del *origen* al *destino*

Mover N-1 discos del *auxiliar* al *destino*

Mover 4 discos de A a C



Luis Hernández Yáñez



# Las Torres de Hanoi

Mover N-1 discos se hace igual, pero usando ahora otros origen y destino

Mover N-1 discos del pilar A al pilar B:

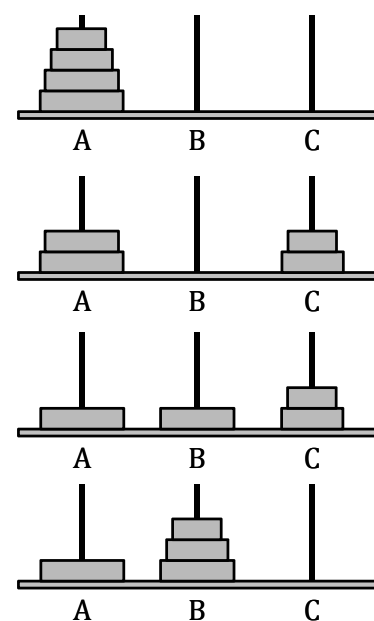
Mover N-2 discos del pilar A al pilar C

Mover el disco del pilar A al pilar B

Mover N-2 discos del pilar C al pilar B

Naturaleza recursiva de la solución

Mover 3 discos de A a B



Luis Hernández Yáñez



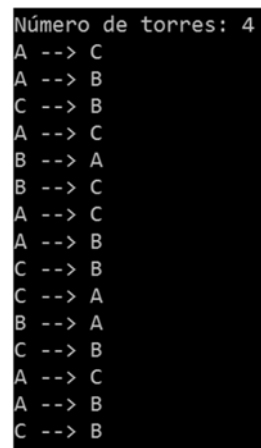
Simulación para 4 discos (wikipedia.org)

Caso base: no quedan discos que mover

```
...
void hanoi(int n, char origen, char destino, char auxiliar) {
    if (n > 0) {
        hanoi(n - 1, origen, auxiliar, destino);
        cout << origen << " --> " << destino << endl;
        hanoi(n - 1, auxiliar, destino, origen);
    }
}

int main() {
    int n;
    cout << "Número de torres: ";
    cin >> n;
    hanoi(n, 'A', 'C', 'B');

    return 0;
}
```



## Fundamentos de la programación

# Recursión frente a iteración



# Recursión frente a iteración

---

```
long long int factorial(int n) {
    long long int fact;

    assert(n >= 0);

    if (n == 0) {
        fact = 1;
    }
    else {
        fact = n * factorial(n - 1);
    }

    return fact;
}
```

```
long long int factorial(int n) {
    long long int fact = 1;

    assert(n >= 0);

    for (int i = 1; i <= n; i++) {
        fact = fact * i;
    }

    return fact;
}
```



# Recursión frente a iteración

---

## *¿Qué es preferible?*

Cualquier algoritmo recursivo tiene uno iterativo equivalente

Los recursivos son menos eficientes que los iterativos:

- Sobrecarga de las llamadas a subprograma

Si hay una versión iterativa sencilla, será preferible a la recursiva

En ocasiones la versión recursiva es mucho más simple

- Será preferible si no hay requisitos de rendimiento

Compara las versiones recursivas del factorial o de los números de Fibonacci con sus equivalentes iterativas

*¿Y qué tal una versión iterativa para los números de Ackermann?*



## Estructuras de datos recursivas



## Estructuras de datos recursivas

---

### *Definición recursiva de listas*

Ya hemos definido de forma recursiva alguna estructura de datos:

Secuencia  $\left\{ \begin{array}{l} \text{elemento seguido de una secuencia} \\ \text{secuencia vacía (ningún elemento)} \end{array} \right.$

Las listas son secuencias:

Lista  $\left\{ \begin{array}{l} \text{elemento seguido de una lista} \\ \text{lista vacía (ningún elemento)} \end{array} \right.$  (Caso base)

La lista 1, 2, 3 consiste en el elemento 1 seguido de la lista 2, 3, que, a su vez, consiste en el elemento 2 seguido de la lista 3, que, a su vez, consiste en el elemento 3 seguido de la lista vacía (caso base)

Hay otras estructuras con naturaleza recursiva (p.e., los árboles) que estudiarás en posteriores cursos



# Estructuras de datos recursivas

---

## *Procesamiento de estructuras de datos recursivas*

Naturaleza recursiva de las estructuras: procesamiento recursivo

*Procesar (lista):*

*Si lista no vacía (caso base):*

*Procesar el primer elemento de la lista // Código anterior*

*Procesar (resto(lista))*

*Procesar el primer elemento de la lista // Código posterior*

*resto(lista):* sublista tras quitar el primer elemento



## Acerca de *Creative Commons*




---



### *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.







## Apéndice: Archivos binarios

Grado en Ingeniería Informática  
Grado en Ingeniería del Software  
Grado en Ingeniería de Computadores

Luis Hernández Yáñez  
Facultad de Informática  
Universidad Complutense



## Índice

---

Flujos	1051
Archivos binarios	1054
Tamaño de los datos: El operador sizeof()	1056
Apertura de archivos binarios	1059
Lectura de archivos binarios (acceso secuencial)	1061
Escritura en archivos binarios (acceso secuencial)	1066
Acceso directo o aleatorio	1070
Ejemplos de uso de archivos binarios	1078
Ordenación de los registros del archivo	1079
Búsqueda binaria	1085
Inserción en un archivo binario ordenado	1088
Carga de los registros de un archivo en una tabla	1092
Almacenamiento de una tabla en un archivo	1093



## Flujos



## Entrada/salida

---

### *Flujos*

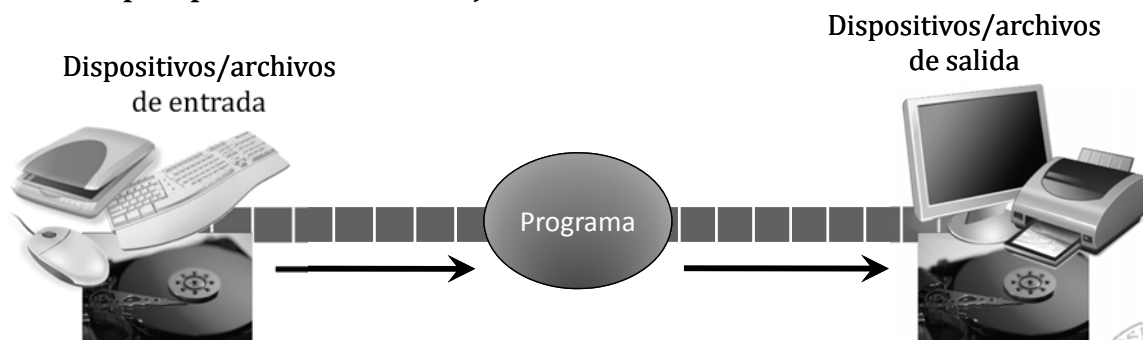
Canalizan la E/S entre los dispositivos y el programa

En forma de secuencias de caracteres

La entrada puede proceder de un dispositivo o de un archivo

La salida puede dirigirse a un dispositivo o a un archivo

Siempre por medio de flujos



# Flujos

---

## *Flujos de texto y binarios*

- ✓ Flujo de texto: contiene una secuencia de caracteres

T	o	t	a	l	:		1	2	3	.	4	↵	A	...
---	---	---	---	---	---	--	---	---	---	---	---	---	---	-----

- ✓ Flujo binario: contiene una secuencia de códigos binarios.

A0	25	2F	04	D6	FF	00	27	6C	CA	49	07	5F	A4	...
----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----

(Códigos representados en notación hexadecimal.)

Lo que signifiquen los códigos dependerá del programa que use el archivo

En ambos casos se trata de una secuencia de caracteres

En el segundo caso se interpretan como códigos binarios

Sin contemplar caracteres especiales como \n o \t

Ya hemos usado flujos de texto para E/S por consola/archivos



---

# Fundamentos de la programación

## Archivos binarios



# Archivos

## Codificación textual y binaria

Datos numéricos: se pueden guardar en forma textual o binaria

```
int dato = 124567894;
```

Representación como texto: caracteres '1' '2' '4' '5' '6' ...

Flujo de texto

1	2	4	5	6	7	8	9	4		
---	---	---	---	---	---	---	---	---	--	--

9 caracteres (se guarda el código ASCII de cada uno)

Representación binaria:

```
00000111 01101100 11000001 01010110 Hex: 07 6C C1 56
```

Flujo binario

07	6C	C1	56							
----	----	----	----	--	--	--	--	--	--	--

4 caracteres interpretados como códigos binarios



# Archivos binarios

## El operador sizeof()

En los archivos binarios se manejan códigos binarios (bytes)

sizeof() (palabra clave): bytes que ocupa en memoria algo

Se aplica a un dato o a un tipo

char ≡ byte

```
const int Max = 80;
```

```
typedef char tCadena[Max];
```

```
typedef struct {
```

```
    int codigo;
```

```
    tCadena item;
```

```
    double valor;
```

```
} tRegistro;
```

```
const int SIZE = sizeof(tRegistro);
```

En un archivo binario un dato del tipo tRegistro

ocupará exactamente SIZE *caracteres*



# El operador sizeof()

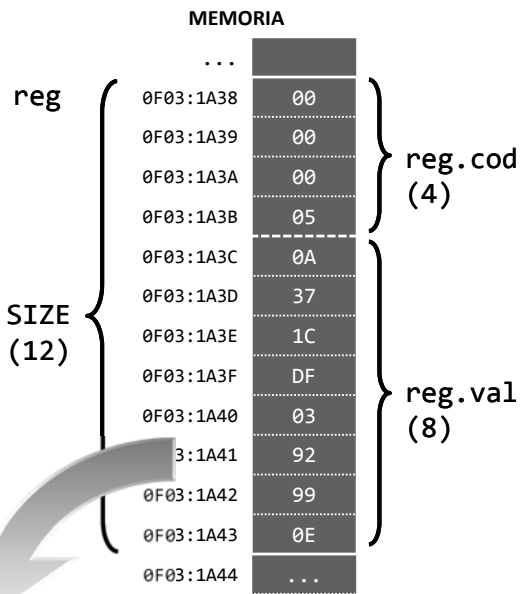
```
typedef struct {
    int cod;
    double val;
} tRegistro;
tRegistro reg;
const int SIZE = sizeof(reg);
...
```

Posiciones de memoria usadas →

Se guardan los SIZE bytes:

Flujo binario

00 00 00 05 0A 37 1C DF 03 92 99 0E



## Alineamiento a tamaño de palabra

Por eficiencia, algunos campos de una estructura se pueden forzar a ocupar un múltiplo del tamaño de palabra del sistema  
Tamaño de palabra (4, 8, 16, ... bytes): dato más pequeño que se lee de la memoria (aunque se usen sólo algunos de los bytes)  
Así, el tamaño real de las estructuras puede ser mayor que la simple suma de los tamaños de cada tipo

Por ejemplo:

```
typedef struct {
    char c;
    int i;
} tRegistro;
const int SIZE = sizeof(tRegistro);
```

char (1 byte) + int (4 bytes)    SIZE toma el valor 8 (4 + 4), no 5

char + int + double            → 24 bytes (8 + 8 + 8)

NOTA: El tamaño de palabra y los tamaños de los tipos dependen del sistema concreto



## Apertura de archivos binarios



## Apertura de archivos binarios

Biblioteca fstream

Archivos binarios: tipo `fstream`

Apertura: función `open(nombre, modo)`

Nombre: `char[]` (función `c_str()` para cadenas de tipo `string`)

Modos de apertura del archivo:

Modo	Significado
<code>ios::app</code>	<i>Añadir</i> : permite seguir escribiendo a partir del final
<code>ios::binary</code>	<i>Binario</i> : tratar el archivo como archivo binario
<code>ios::in</code>	<i>Entrada</i> : archivo para leer de él
<code>ios::out</code>	<i>Salida</i> : archivo para escribir en él
<code>ios::trunc</code>	<i>Truncar</i> : borrar todo lo que haya y empezar de nuevo

Concatenación de modos: operador `|` (0 binaria: *suma bit a bit*)

```
archivo.open("entrada.dat", ios::in | ios::binary);
```



## Lectura de archivos binarios (acceso secuencial)



## Lectura de archivos binarios

---

`archivo.read(puntero_al_búfer, número)`

*búfer*: variable destino de los caracteres leídos

Pasado como puntero a secuencia de caracteres

Referencia (&) a la variable destino

Molde de puntero a carácter (`char *`)

*número*: cantidad de caracteres a extraer del archivo

→ Operador `sizeof()`

Archivo abierto con los modos `ios::in` e `ios::binary`

`archivo.read( (char *) &registro, sizeof(tRegistro));`

Los caracteres leídos se interpretan como códigos binarios



# Lectura de archivos binarios

## Éxito o fallo de la lectura

Función gcount()

Nº de caracteres realmente leídos en la última operación

Si coincide con el número que se solicitaron leer: OK

Si son menos, se ha alcanzado el final del archivo: Fallo

```
tRegistro registro;
fstream archivo;
archivo.open("entrada.dat", ios::in | ios::binary);
archivo.read( (char *) &registro, sizeof(tRegistro));
if (archivo.gcount() < sizeof(tRegistro)) {
    // Fallo en la lectura
}
else {
    // Lectura OK
    ...
}
```

Luis Hernández Yáñez



# Lectura de archivos binarios

leer.cpp

```
#include <iostream>
using namespace std;
#include <fstream>
#include "registro.h"

int main() {
    tRegistro registro;
    fstream archivo;
    archivo.open("bd.dat", ios::in | ios::binary);
    archivo.read( (char *) &registro, SIZE);
    int cuantos = archivo.gcount();
    while (cuantos == SIZE) {
        mostrar(registro);
        archivo.read( (char *) &registro, SIZE);
        cuantos = archivo.gcount();
    }
    archivo.close(); ← ¡No olvides cerrar el archivo!
    return 0;
}
```

12345	- Disco duro	- 123.59 euros
324356	- Placa base core i7	- 234.50 euros
2121	- Multipuerto USB	- 15.00 euros
54354	- Disco externo 500 Gb	- 97.80 euros
112341	- Procesador AMD	- 132.95 euros
66678325	- Marco digital 2 Gb	- 78.99 euros
600673	- Monitor 22" Nisu	- 154.50 euros
11111	- Disco externo 1 Tb	- 125.95 euros

Luis Hernández Yáñez





## El tipo tRegistro

```
const int Max = 80;
typedef char tCadena[Max];
typedef struct {
    int codigo;
    tCadena item;
    double valor;
} tRegistro;
const int SIZE = sizeof(tRegistro);
```

¿Por qué usamos cadenas al estilo de C?

string: tamaño variable en memoria

Requieren un proceso de *serialización*

Las cadenas al estilo de C siempre ocupan lo mismo en memoria



# Fundamentos de la programación

## Escritura en archivos binarios (acceso secuencial)



# Escritura en archivos binarios

---

`archivo.write(puntero_al_búfer, número)`

*búfer*: origen de los caracteres a escribir en el archivo

Pasado como puntero a secuencia de caracteres

Referencia (&) a la variable destino

Molde de puntero a carácter (`char *`)

*número*: cantidad de caracteres a escribir en el archivo

→ Operador `sizeof()`

Archivo abierto con los modos `ios::out` e `ios::binary`

```
archivo.write( (char *) &registro, sizeof(tRegistro));
```

Se escriben tantos caracteres como celdas de memoria ocupe la variable `registro`



# Escritura en archivos binarios

---

`escribir.cpp`

```
#include <iostream>
using namespace std;
#include <fstream>
#include "registro.h"

int main() {
    tRegistro registro;
    fstream archivo;
    archivo.open("bd2.dat", ios::out | ios::binary);
    bool seguir = true;
    while (seguir) {
        cout << "Código: ";
        cin.sync();
        cin >> registro.codigo;
        cout << "Nombre: ";
        cin.sync();
        cin.getline(registro.item, Max); // Máx: 80
        ...
    }
}
```



# Escritura en archivos binarios

---

```
    cout << "Precio: ";
    cin.sync();
    cin >> registro.valor;
    archivo.write( (char *) &registro, SIZE);
    cout << "Otro [S/N]? ";
    char c;
    cin >> c;
    if ((c == 'n') || (c == 'N')) {
        seguir = false;
    }
}
archivo.close(); ← ¡No olvides cerrar el archivo!
                    (¡pérdida de datos!)

return 0;
}
```



# Fundamentos de la programación

---

## Acceso directo o aleatorio



# Archivos binarios: acceso directo

Acceso secuencial: empezando en el primero pasando a siguiente  
Acceso directo (también llamado aleatorio):

Para localizar registros individuales necesitamos otras rutinas:

✓ `tellg()`: lugar donde se encuentra el puntero del archivo  
Siguiete posición donde se realizará una lectura o escritura

✓ `seekg(desplazamiento, origen)`:

Lleva el puntero del archivo a una posición concreta:  
*desplazamiento* caracteres desde el *origen* indicado

Origen:

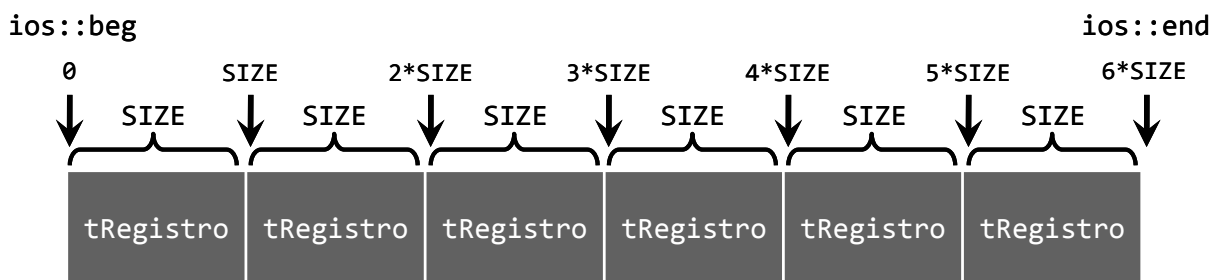
`ios::beg`: principio del archivo

`ios::cur`: posición actual

`ios::end`: final del archivo



# Archivos binarios: acceso directo



```
const int SIZE = sizeof(tRegistro);
```

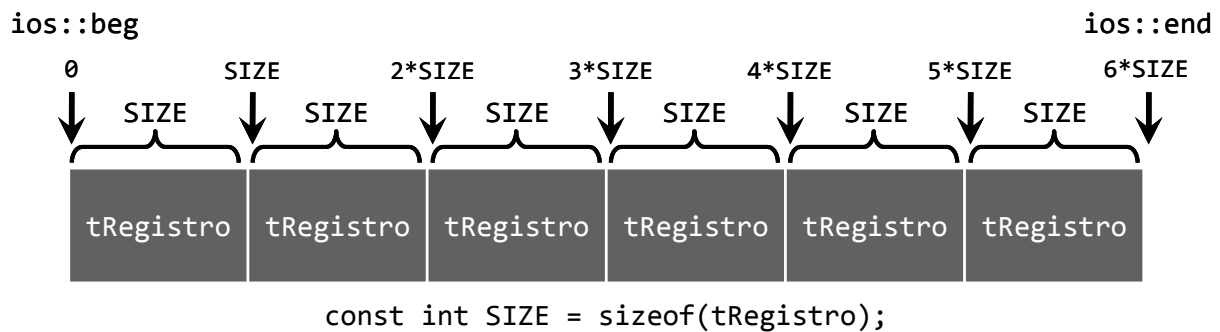
Cada registro ocupa `SIZE` caracteres en el archivo

¿Cuántos registros hay en el archivo?

```
archivo.seekg(0, ios::end); // 0 car. desde el final -> final
int pos = archivo.tellg(); // Total de caracteres del archivo
int numReg = pos / SIZE;
```



# Archivos binarios: acceso directo



Poner el puntero del archivo en un nº de registro:

```
archivo.seekg((num - 1) * SIZE, ios::beg);
```



# Archivos binarios: acceso directo

## *Lecturas y escrituras*

Una vez ubicado el puntero al principio de un registro, se puede leer el registro o actualizar (escribir) el registro

Si se ubica al final, se puede añadir (escribir) un nuevo registro

Archivos binarios de lectura/escritura:

Se han de abrir con los modos `ios::in`, `ios::out` e `ios::binary`  
`archivo.open("bd.dat", ios::in | ios::out | ios::binary);`

Ahora podemos tanto leer como escribir



```
// Actualización de un registro
#include <iostream>
using namespace std;
#include <fstream>
#include "registro.h"

int main() {
    tRegistro registro;
    fstream archivo;

    archivo.open("bd.dat", ios::in | ios::out | ios::binary);
    archivo.seekg(0, ios::end);
    int pos = archivo.tellg();
    int numReg = pos / SIZE;
    cout << "Número de registros: " << numReg << endl;
    int num;
    cout << "Registro número? ";
    cin >> num;
    ...
}
```

Luis Hernández Yáñez



# Acceso directo

```
if ((num > 0) && (num <= numReg)) {
    archivo.seekg((num - 1) * SIZE, ios::beg);
    archivo.read( (char *) &registro, SIZE);
    mostrar(registro);
    cout << endl << "Cambiar nombre [S/N]? ";
    char c;
    cin.sync();
    cin >> c;
    if ((c == 's') || (c == 'S')) {
        cout << "Nombre: ";
        cin.sync();
        cin.getline(registro.item, 80);
    }
    cout << endl << "Cambiar precio [S/N]? ";
    cin.sync();
    cin >> c;
    if ((c == 's') || (c == 'S')) {
        cout << "Precio: ";
        cin >> registro.valor;
    }
    ...
}
```

Luis Hernández Yáñez



# Acceso directo

---

```
    archivo.seekg((num - 1) * SIZE, ios::beg);
    archivo.write( (char *) &registro, SIZE);
    cout << endl << "Registro actualizado:" << endl;
    archivo.seekg((num - 1) * SIZE, ios::beg);
    archivo.read( (char *) &registro, SIZE);
    mostrar(registro);
}
archivo.close();
return 0;
}
```

```
Número de registros: 8
Registro número? 4
    54354 - Disco externo 500 Gb          -    97.80 euros

Cambiar nombre [S/N]? n

Cambiar precio [S/N]? s
Precio: 98.5

Registro actualizado:
    54354 - Disco externo 500 Gb          -    98.50 euros
```



# Fundamentos de la programación

---

## Ejemplos de uso de archivos binarios



Mediante un acceso directo a los registros del archivo  
Ordenaremos por el campo `item`

```
#include <iostream>
using namespace std;
#include <fstream>
#include <iomanip>
#include <cstring>
#include "registro.h"

const char BD[] = "lista.dat";

void mostrar();

...
```



# Ordenación de los registros

```
void mostrar() {
    fstream archivo;
    tRegistro registro;
    int cuantos;

    archivo.open(BD, ios::in | ios::binary);
    archivo.read( (char *) &registro, SIZE);
    cuantos = archivo.gcount();
    while (cuantos == SIZE) {
        mostrar(registro);
        archivo.read( (char *) &registro, SIZE);
        cuantos = archivo.gcount();
    }
    archivo.close();
}

...
```





# Ordenación de los registros

```
int main() {  
    mostrar();
```

Orden inicial

12345	- Memoria 4Gb	-	120.95 euros
13427	- Memoria USB 2Gb	-	32.50 euros
21115	- Multipuerto USB 4x1	-	12.95 euros
33241	- Memoria 8Gb	-	243.99 euros
37253	- Placa base core i7	-	178.95 euros
52236	- Multidrive DVD-X10	-	39.50 euros
57890	- Conversor VGA-DVI	-	19.95 euros
66638	- Unidad Blue-Ray	-	54.95 euros
71934	- Placa base core i3	-	125.00 euros
84956	- Memoria 2Gb	-	76.96 euros

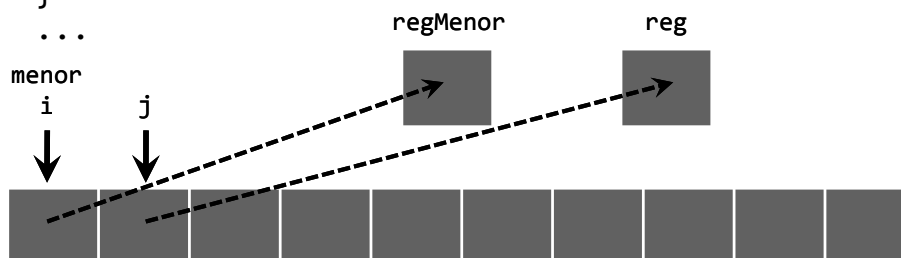
```
fstream archivo;  
archivo.open(BD, ios::in | ios::out | ios::binary);  
archivo.seekg(0, ios::end);  
int pos = archivo.tellg();  
int numReg = pos / SIZE;  
...
```

Luis Hernández Yáñez



# Ordenación de los registros

```
// Ordenamos con el método de selección directa  
tRegistro regMenor, reg;  
for (int i = 0; i < numReg - 1; i++) {  
    int menor = i;  
    for (int j = i + 1; j < numReg; j++) {  
        archivo.seekg(menor * SIZE, ios::beg);  
        archivo.read( (char *) &regMenor, SIZE);  
        archivo.seekg(j * SIZE, ios::beg);  
        archivo.read( (char *) &reg, SIZE);  
        if (strcmp(reg.item, regMenor.item) < 0) {  
            menor = j;  
        }  
    }  
}
```



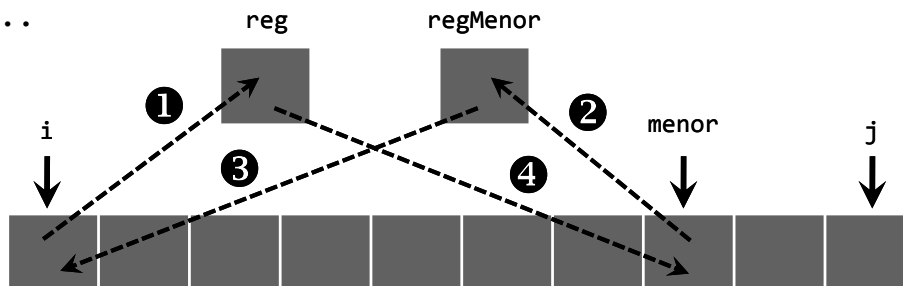
Luis Hernández Yáñez



# Ordenación de los registros

```
if (menor > i) { // Intercambiamos
    archivo.seekg(i * SIZE, ios::beg);
    archivo.read( (char *) &reg, SIZE);
    archivo.seekg(menor * SIZE, ios::beg);
    archivo.read( (char *) &regMenor, SIZE);
    archivo.seekg(i * SIZE, ios::beg);
    archivo.write( (char *) &regMenor, SIZE);
    archivo.seekg(menor * SIZE, ios::beg);
    archivo.write( (char *) &reg, SIZE);
}
}
...

```



Luis Hernández Yáñez



# Ordenación de los registros

```
archivo.close();

cout << endl << "Tras ordenar:" << endl << endl;
mostrar();

return 0;
}

```

```
Tras ordenar:
57890 - Conversor VGA-DVI - 19.95 euros
84956 - Memoria 2Gb - 76.96 euros
12345 - Memoria 4Gb - 120.95 euros
33241 - Memoria 8Gb - 243.99 euros
13427 - Memoria USB 2Gb - 32.50 euros
52236 - Multidrive DVD-X10 - 39.50 euros
21115 - Multipuerto USB 4x1 - 12.95 euros
71934 - Placa base core i3 - 125.00 euros
37253 - Placa base core i7 - 178.95 euros
66638 - Unidad Blue-Ray - 54.95 euros

```

Luis Hernández Yáñez



Archivo binario ordenado; por código

```
#include <iostream>
using namespace std;
#include <fstream>
#include "registro.h"

const char BD[] = "ord.dat";

void mostrar();

int main() {
    mostrar();
    tRegistro registro;
    fstream archivo;
    ...
}
```



# Búsqueda binaria

```
archivo.open(BD, ios::in | ios::binary);
archivo.seekg(0, ios::end);
int pos = archivo.tellg();
int numReg = pos / SIZE;
int buscado;
cout << "Código a buscar: ";
cin >> buscado;
int ini = 0, fin = numReg - 1, mitad;
bool encontrado = false;
while ((ini <= fin) && !encontrado) {
    mitad = (ini + fin) / 2;
    archivo.seekg(mitad * SIZE, ios::beg);
    archivo.read( (char *) &registro, SIZE);
    if (buscado == registro.codigo) {
        encontrado = true;
    }
    else if (buscado < registro.codigo) {
        fin = mitad - 1;
    }
    ...
}
```



# Búsqueda binaria

---

```
        else {
            ini = mitad + 1;
        }
    }
    if (encontrado) {
        int pos = mitad + 1;
        cout << "Encontrado en la posición " << pos << endl;
        mostrar(registro);
    }
    else {
        cout << "No encontrado!" << endl;
    }
    archivo.close();

    return 0;
}

...

```

Luis Hernández Yáñez



# Inserción en un archivo ordenado

---

insertar.cpp

Ordenado por el campo código

```
#include <iostream>
using namespace std;
#include <fstream>
#include "registro.h"

const char BD[] = "ord2.dat";

void mostrar();

int main() {
    mostrar();
    tRegistro nuevoRegistro = nuevo(), registro;
    fstream archivo;
    archivo.open(BD, ios::in | ios::out | ios::binary);
    archivo.seekg(0, ios::end);
    int pos = archivo.tellg();
    int numReg = pos / SIZE;
    ...
}

```

Luis Hernández Yáñez



# Inserción en un archivo ordenado

```
pos = 0;
bool encontrado = false;
archivo.seekg(0, ios::beg);
while ((pos < numReg) && !encontrado) {
    archivo.read( (char *) &registro, SIZE);
    if (registro.codigo > nuevoRegistro.codigo) {
        encontrado = true;
    }
    else {
        pos++;
    }
}
if (pos == numReg) { // Debe ir al final
    archivo.seekg(0, ios::end);
    archivo.write( (char *) &nuevoRegistro, SIZE);
}
...
```

Luis Hernández Yáñez

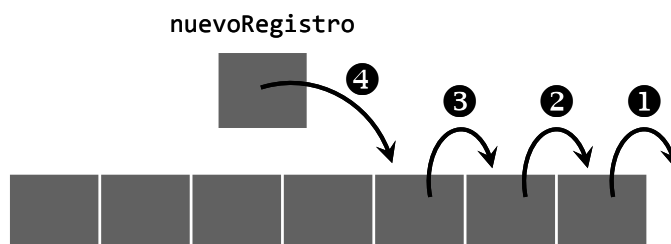


# Inserción en un archivo ordenado

```
else { // Hay que hacer hueco
    for (int i = numReg - 1; i >= pos; i--) {
        archivo.seekg(i * SIZE, ios::beg);
        archivo.read( (char *) &registro, SIZE);
        archivo.seekg((i + 1) * SIZE, ios::beg);
        archivo.write( (char *) &registro, SIZE);
    }
    archivo.seekg(pos * SIZE, ios::beg);
    archivo.write( (char *) &nuevoRegistro, SIZE);
}
archivo.close();

mostrar();

return 0;
}
```



Luis Hernández Yáñez



# Inserción en un archivo ordenado

```
12345 - Memoria 4Gb - 120.95 euros
13427 - Memoria USB 2Gb - 32.50 euros
21115 - Multipuerto USB 4x1 - 12.95 euros
37253 - Placa base core i7 - 178.95 euros
57890 - Conversor VGA-DVI - 19.95 euros
```

Nuevo registro:

Código: 11111

Nombre: Memoria 8Gb

Precio: 189

```
11111 - Memoria 8Gb
12345 - Memoria 4Gb
13427 - Memoria USB 2Gb
21115 - Multipuerto USB 4x1
37253 - Placa base core i7
57890 - Conversor VGA-DVI
```

Al principio

```
12345 - Memoria 4Gb - 120.95 euros
13427 - Memoria USB 2Gb - 32.50 euros
21115 - Multipuerto USB 4x1 - 12.95 euros
37253 - Placa base core i7 - 178.95 euros
57890 - Conversor VGA-DVI - 19.95 euros
```

Nuevo registro:

Código: 22222

Nombre: Memoria 8Gb

Precio: 189

```
12345 - Memoria 4Gb
13427 - Memoria USB 2Gb
21115 - Multipuerto USB 4x1
22222 - Memoria 8Gb
37253 - Placa base core i7
57890 - Conversor VGA-DVI
```

Por el medio

```
12345 - Memoria 4Gb - 120.95 euros
13427 - Memoria USB 2Gb - 32.50 euros
21115 - Multipuerto USB 4x1 - 12.95 euros
37253 - Placa base core i7 - 178.95 euros
57890 - Conversor VGA-DVI - 19.95 euros
```

Nuevo registro:

Código: 66666

Nombre: Memoria 8Gb

Precio: 189

```
12345 - Memoria 4Gb
13427 - Memoria USB 2Gb
21115 - Multipuerto USB 4x1
37253 - Placa base core i7
57890 - Conversor VGA-DVI
66666 - Memoria 8Gb
```

Al final



## Carga de los registros en una tabla

tabla.cpp

```
void cargar(tTabla &tabla, bool &ok) {
    ok = true;
    fstream archivo;
    archivo.open(BD, ios::in | ios::binary);
    if (!archivo.is_open()) {
        ok = false;
    }
    else {
        archivo.seekg(0, ios::end);
        int pos = archivo.tellg();
        int numReg = pos / SIZE;
        tabla.cont = 0;
        tRegistro registro;
        archivo.seekg(0, ios::beg);
        for (int i = 0; i < numReg; i++) {
            archivo.read( (char *) &registro, SIZE);
            tabla.registros[tabla.cont] = registro;
            tabla.cont++;
        }
        archivo.close();
    }
}
```



```
void guardar(tTabla tabla) {
    fstream archivo;
    archivo.open(BD, ios::out | ios::binary | ios::trunc);
    for (int i = 0; i < tabla.cont; i++) {
        archivo.write( (char *) &tabla.registros[i], SIZE);
    }
    archivo.close();
}
```



## Carga y almacenamiento

```
#include <iostream>
using namespace std;
#include "registro.h"
#include "tabla.h"

int main() {
    tTabla tabla;
    tTabla ok;
    cargar(tabla, ok);
    if (!ok) {
        cout << "Error al abrir el archivo!" << endl;
    }
    else {
        mostrar(tabla);
        insertar(tabla, nuevo(), ok);
        mostrar(tabla);
        guardar(tabla);
    }
    return 0;
}
```








## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  Reconocimiento (*Attribution*):  
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  No comercial (*Non commercial*):  
La explotación de la obra queda limitada a usos no comerciales.
-  Compartir igual (*Share alike*):  
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.

