

## 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  $\equiv$  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*

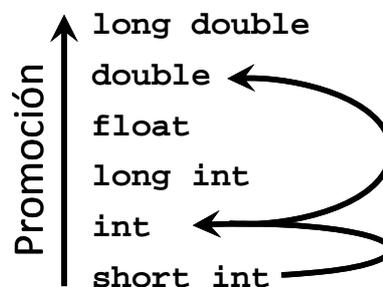
```
short int i = 3;  
int j = 2;  
double a = 1.5, b;  
b = a + i * j;
```

```
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)



Luis Hernández Yáñez



## 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

Luis Hernández Yáñez



# 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

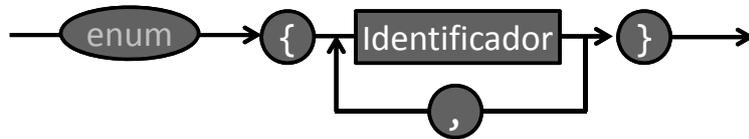
## Tipos enumerados



# 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

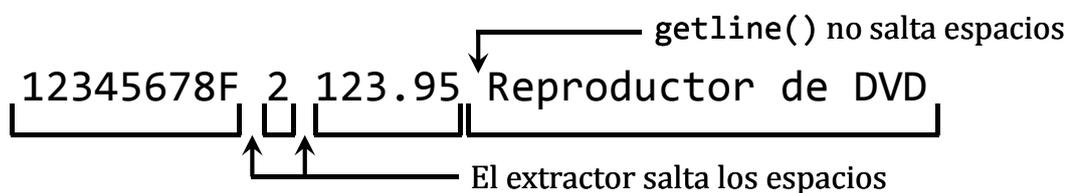


Luis Hernández Yáñez



# Lectura de archivos de texto

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



Luis Hernández Yáñez



# 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



# 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



# 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;
}
...
```



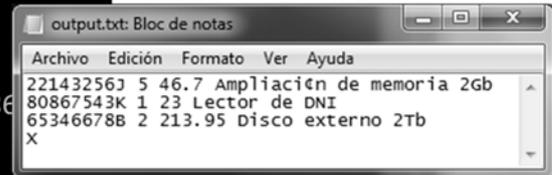
# Escritura en archivos de texto

---

```
// 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
```



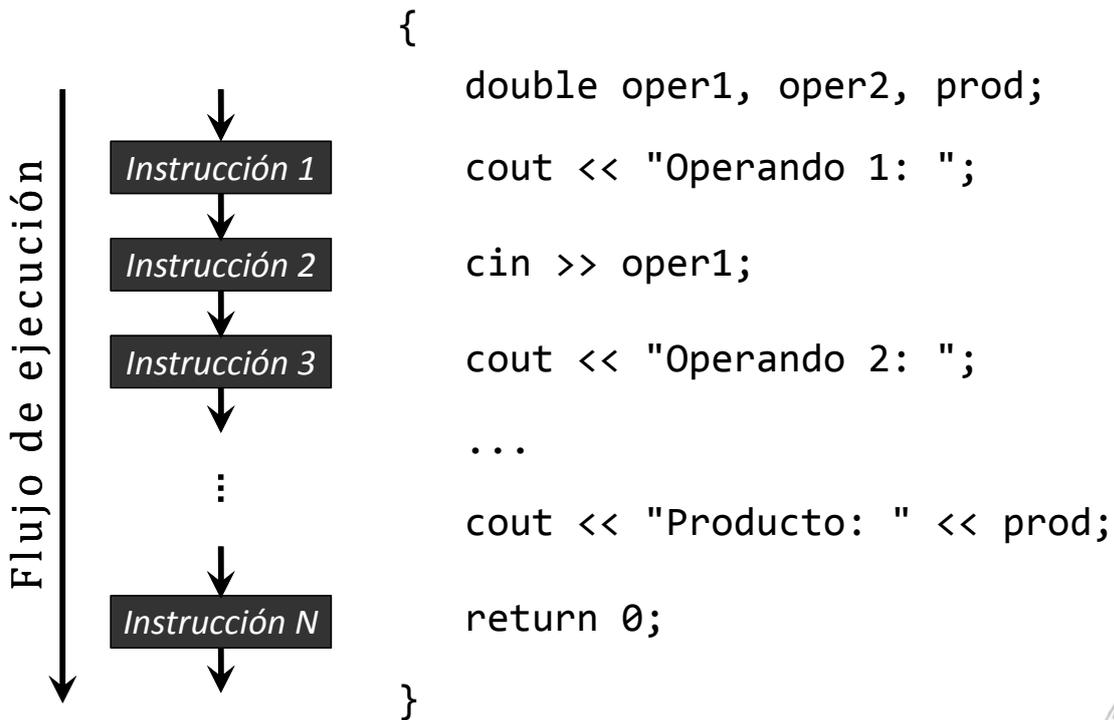
# Fundamentos de la programación

---

## Flujo de ejecución



# Ejecución secuencial



Luis Hernández Yáñez



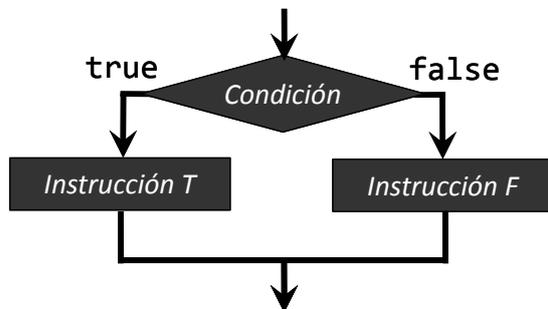
# Selección



*Uno entre dos o más caminos de ejecución*

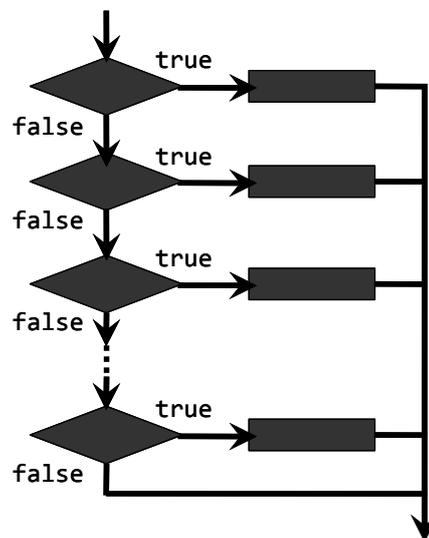
Selección simple (2 caminos)

Selección múltiple (> 2 caminos)



if

if-else-if  
switch



*Diagramas de flujo*

Luis Hernández Yáñez

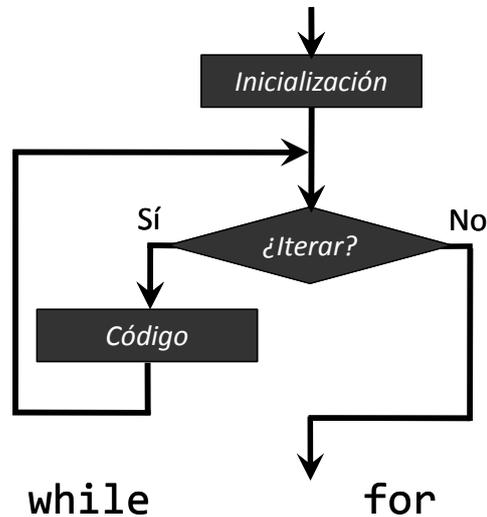


# Repetición (iteración)



*Repetir la ejecución de una o más instrucciones*

Acumular, procesar colecciones, ...



Luis Hernández Yáñez



# Fundamentos de la programación

## Selección simple

Luis Hernández Yáñez



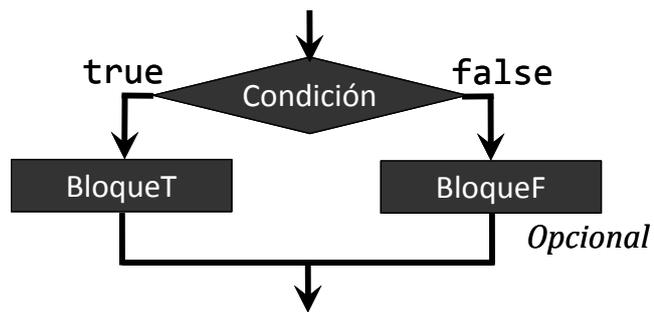
# Selección simple (bifurcación)



## La instrucción if

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

condición: expresión bool  
Cláusula else opcional



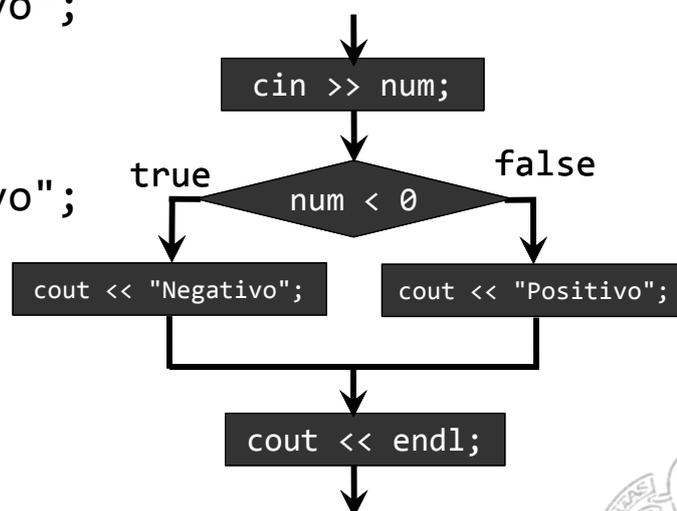
Luis Hernández Yáñez



# La instrucción if

signo.cpp

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



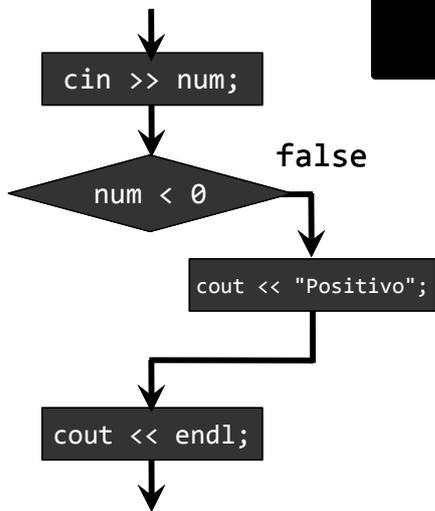
Luis Hernández Yáñez



# La instrucción if

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

```
129  
Positivo
```



num 129

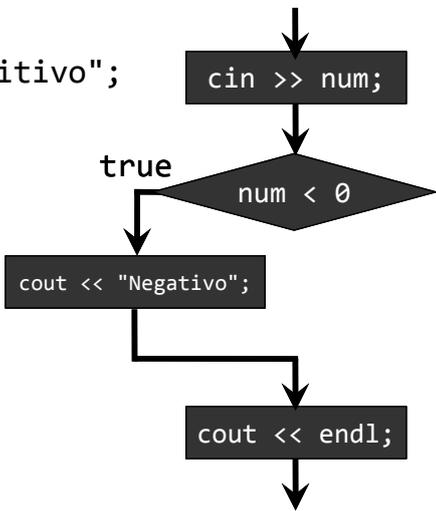
Luis Hernández Yáñez



# 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	true	true	false	true	true	true
false	true	false	false	false	false	true	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))`

Luis Hernández Yáñez



# Fundamentos de la programación

## Anidamiento de if

Luis Hernández Yáñez



```
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!

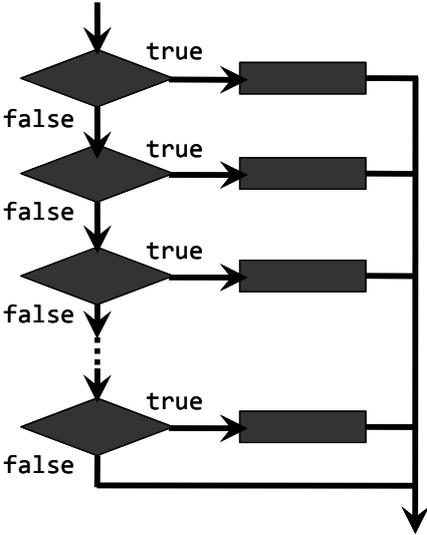


## Selección múltiple

Luis Hernández Yáñez



## Selección múltiple



if-else-if  
switch

Luis Hernández Yáñez



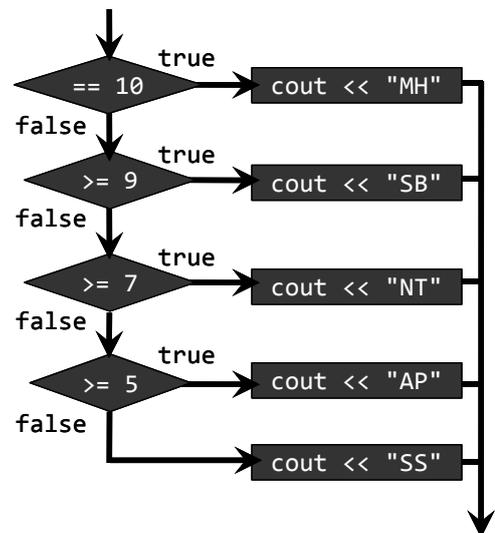
## La escala if-else-if



## 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



# 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



## 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



# ¿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

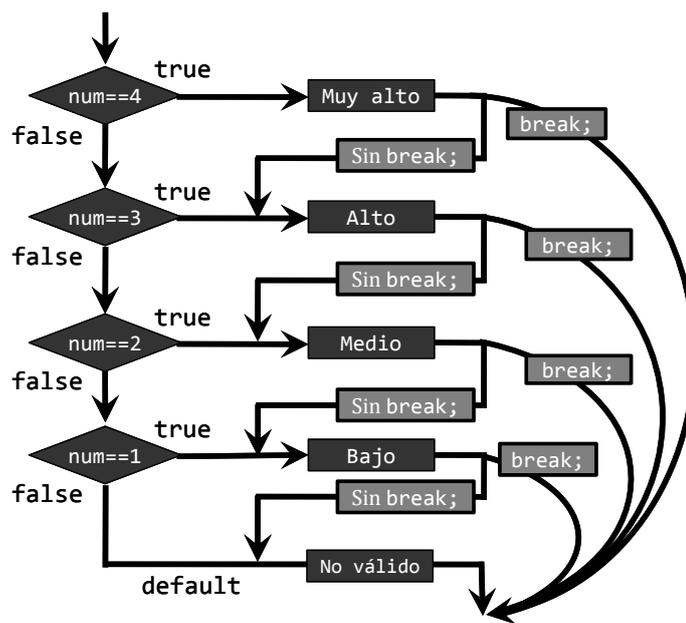
```
switch (num) {  
  ...  
  case ③  
  {  
    cout << "Alto";  
  }  
  case 2:  
  {  
    cout << "Medio";  
  }  
  case 1:  
  {  
    cout << "Bajo";  
  }  
  default:  
  {  
    cout << "Valor no válido";  
  }  
}
```



Luis Hernández Yáñez



# Con y sin break



Luis Hernández Yáñez



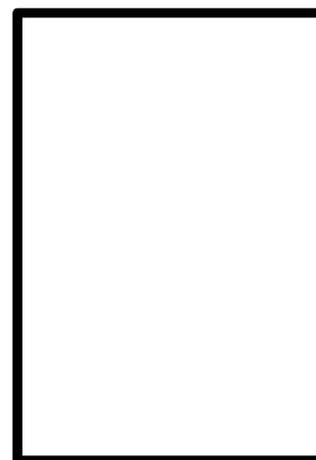
# Un menú

```
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";
    }
}
```

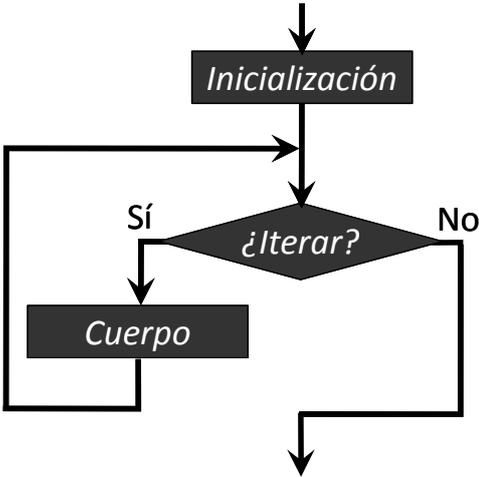


## Repetición

Luis Hernández Yáñez



## Repetición (iteración)



Bucles while y for

Luis Hernández Yáñez



# 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

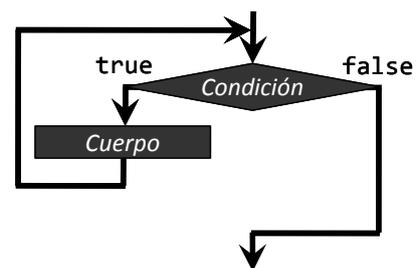
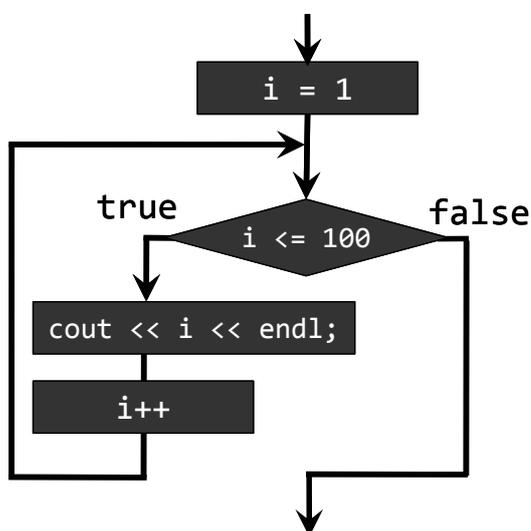
Luis Hernández Yáñez



## Ejecución del bucle while

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

i 101



1
2
3
...
99
100
—

Luis Hernández Yáñez



# 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
```

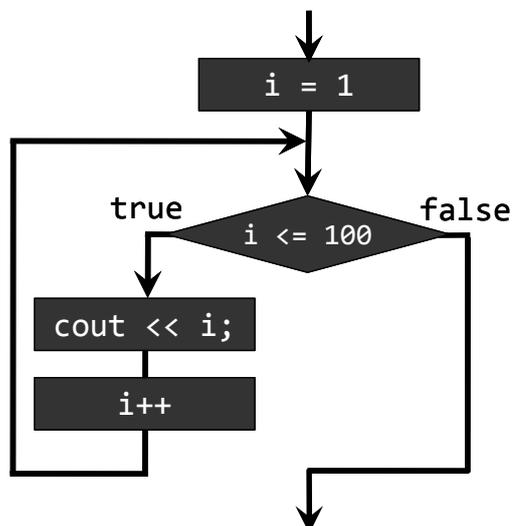
Tantos ciclos como valores toma la variable contadora



# Ejecución del bucle for

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

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

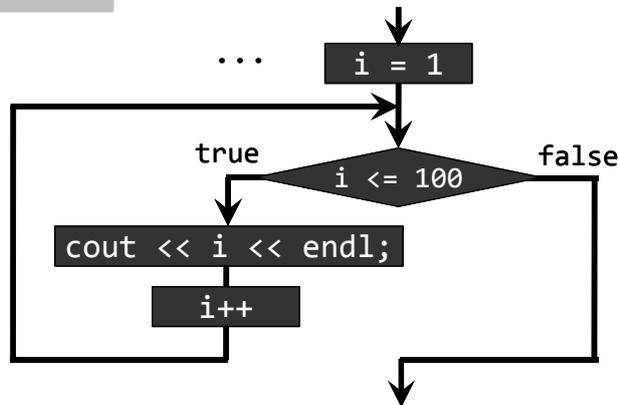


# Ejecución del bucle for

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



Fundamentos de la programación: Tipos e instrucciones II

Página 327



## 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



Fundamentos de la programación: Tipos e instrucciones II

Página 328



# 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

...



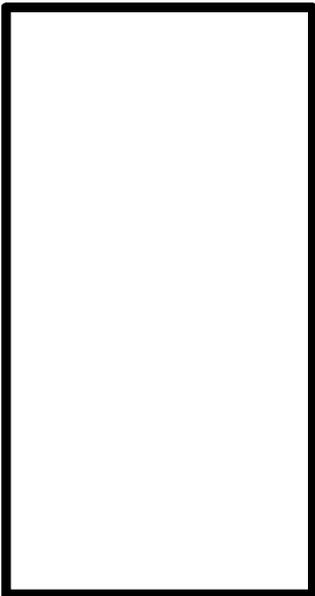
# Tablas de multiplicación

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



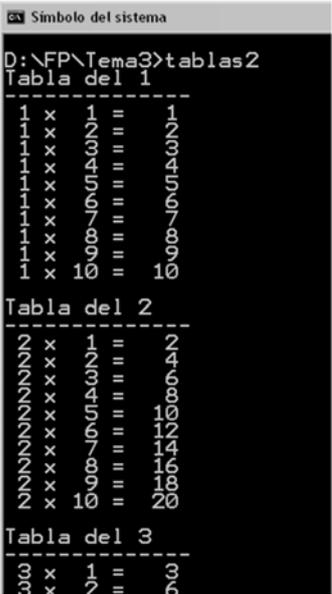
# Mejor presentación

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; ...
        }
    }
}
```

Luis Hernández Yáñez



# 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;
}
```

Luis Hernández Yáñez



# 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++) {
            ...
        }
        Ámbito de la variable 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;  
}
```



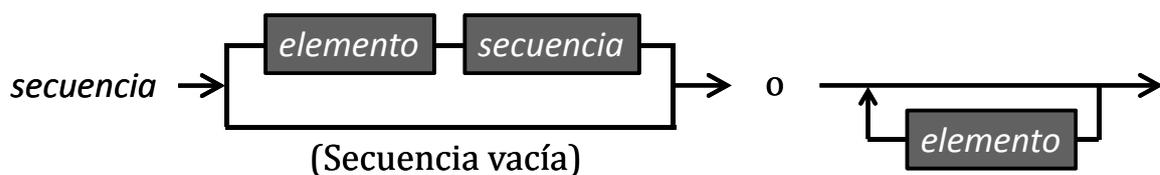
## Secuencias



## 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

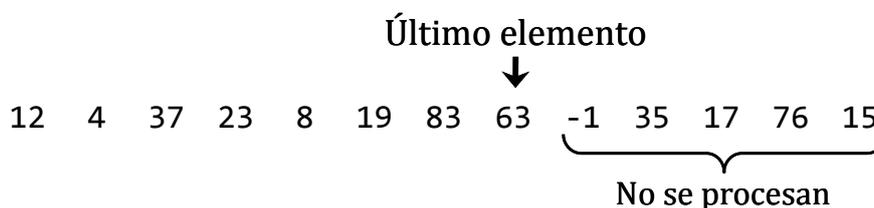
Cadenas de caracteres:

¿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*

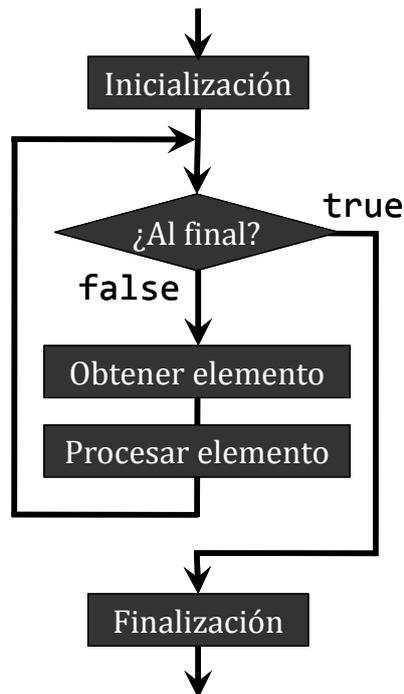
*Finalización*

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*

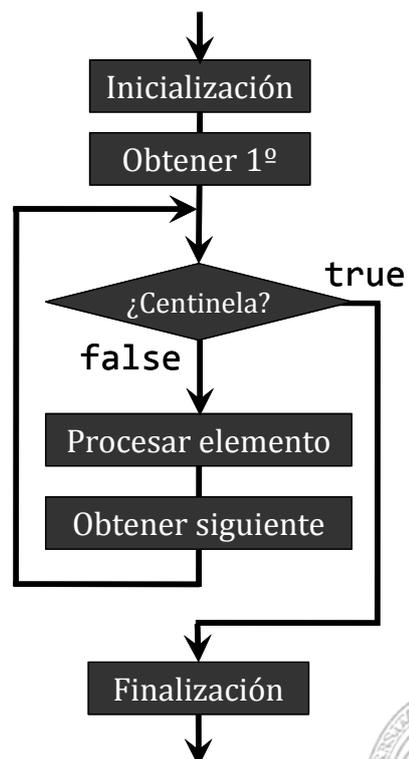
*Obtener el primer elemento*

*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

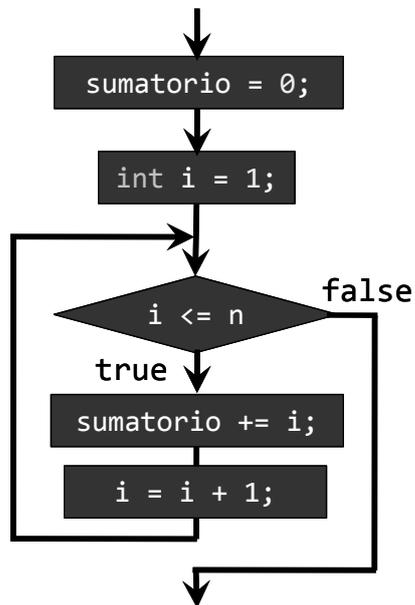
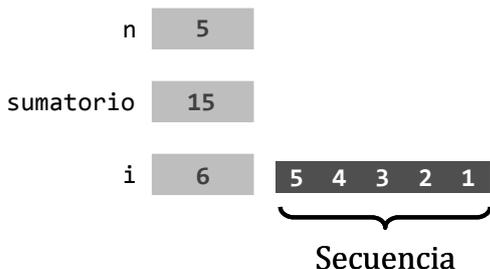
Página 364



# 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

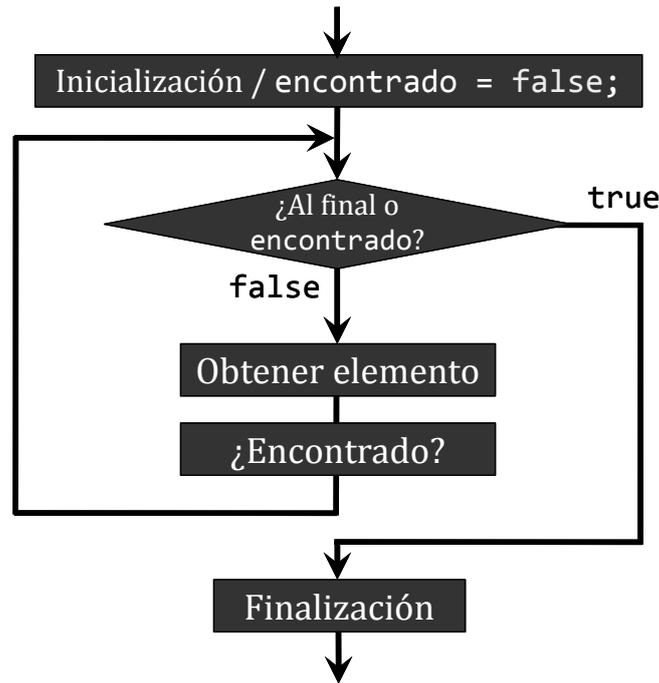
Dos condiciones de terminación del bucle: se encuentra / al final

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

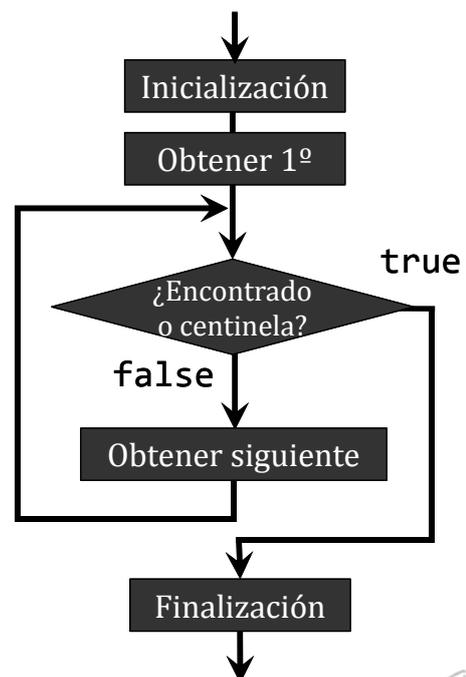
*Inicialización*

*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:

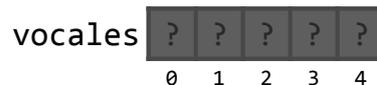
```
tTemp tempMax;
```



```
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;
```

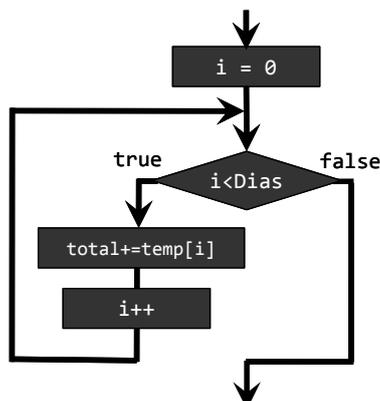
Luis Hernández Yáñez



# 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



# 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



# 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.

