

3

Más sobre tipos e instrucciones

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

Fernando Rosa Velardo
(adaptadas del original de Luis Hernández Yáñez)



Facultad de Informática
Universidad Complutense

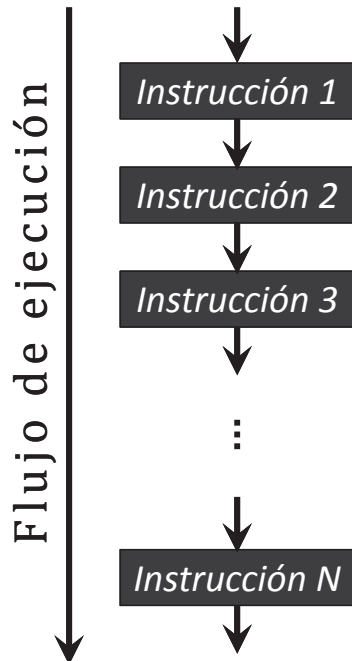


Índice

Flujo de ejecución	2	Tipos de datos	93
La instrucción condicional	5	Conversión de tipos	96
Anidamiento de if	6	Declaración de tipo	99
La instrucción switch	14	Tipos enumerados	100
Repetición	19	Arrays de datos simples	108
El bucle while	23	Uso de variables arrays	112
El bucle do-while	25	Recorrido de arrays	115
El bucle for	29	Búsqueda en arrays	118
Bucles anidados	35	Arrays no completos	121
Ámbito y visibilidad	39	Cadenas de caracteres	127
Archivos de texto	50	Cadenas a la C	131
Lectura de archivos de texto	53		
Escritura en archivos de texto	62		
Secuencias	66		
Recorrido de secuencias	72		
Secuencias calculadas	82		
Búsqueda en secuencias	88		



Ejecución secuencial



Bloques de código

```
{  
    Instrucción 1  
    Instrucción 2  
    Instrucción 3  
    ...  
    Instrucción N  
}
```

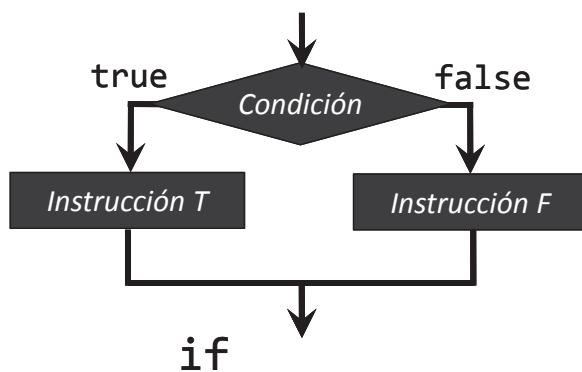


Distinción de casos

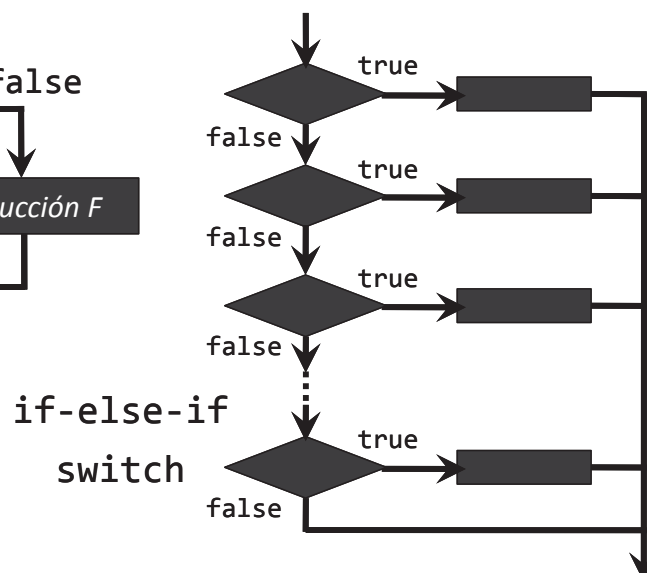


Elegir entre dos o más alternativas de ejecución

Simple (2 caminos)



Múltiple (> 2 caminos)

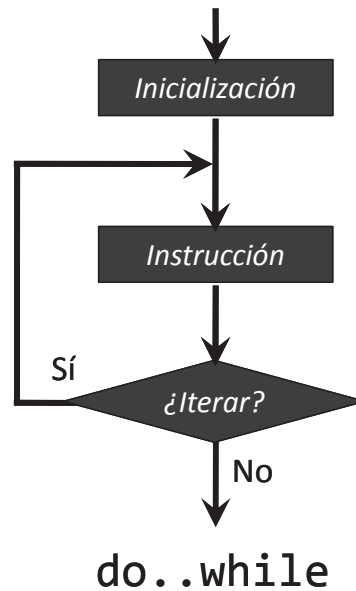
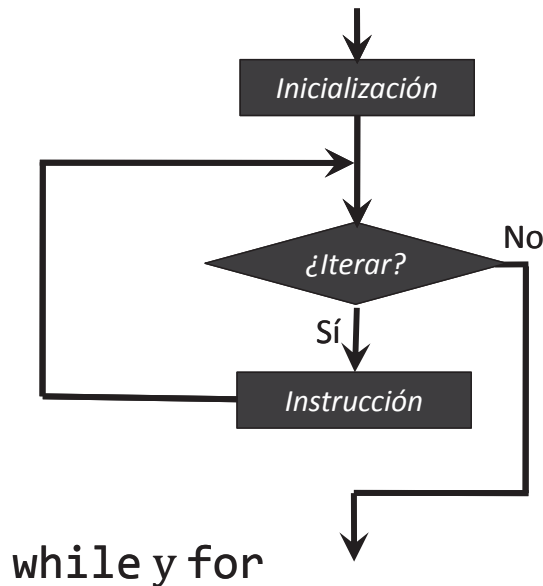


Iteración (Repetición)



Repetir la ejecución de una o más instrucciones

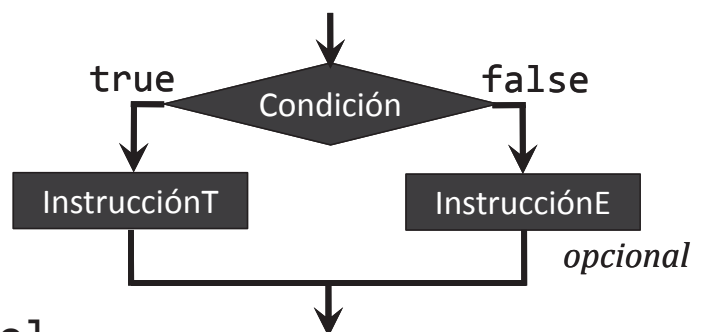
Acumular, procesar colecciones, ...



La instrucción if



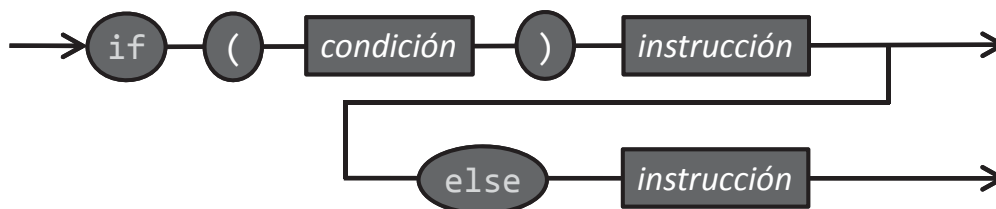
```
if (condición)
    instrucciónT
[else
    instrucciónE]
```



condición: expresión bool

Cláusula else opcional

Instrucciones simples o bloques



Anidamiento de instrucciones if

Número de días del mes y año indicado por el usuario

```
int diasMes(tMes mes, int anio){
    int dias;
    // Distinción de casos
    if (mes == febrero)
        if esBisiesto(mes, anio)
            días = 29;
        else días = 28;
    else if ((mes == abril) || (mes == junio) ||
            (mes == septiembre) || (mes == noviembre))
        días = 30;
    else días = 31;

    return dias;
}
```



Asociación de cláusulas else

Cada else se asocia al if anterior más cercano sin asociar

```
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 ...
```

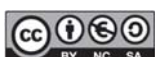
Dos casos distintos:

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

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



La sangría y las llaves de bloque { ... }
ayudan a asociar los else con sus if

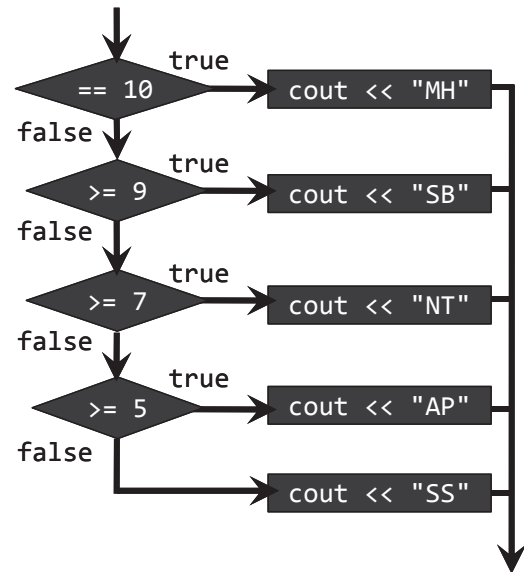


La escala if-else-if

Ejemplo:

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

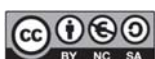
Si nota == 10 entonces MH
si no, si nota >= 9 entonces SB
si no, si nota >= 7 entonces NT
si no, si nota >= 5 entonces AP
si no SS



La escala if-else-if

```
void mostrarNota(unsigned float 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 /*if (nota >= 0)*/  
        cout << "SS";  
}
```

Distinción de casos: las condiciones deben cubrir todos los casos de forma exclusiva.



La escala if-else-if

¡Cuidado con el orden de las condiciones!

```
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";
```



```
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



La escala if-else-if

Simplificación de las condiciones

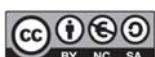


```
if (nota == 10) cout << "MH";  
else if ((nota < 10) && (nota >= 9)) cout << "SB";  
else if ((nota < 9) && (nota >= 7)) cout << "NT";  
else if ((nota < 7) && (nota >= 5)) cout << "AP";  
else if (nota < 5) cout << "SS";
```

```
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



Ejemplo: nivel de un valor

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

```
int main() {
    int num;
    cout << "Introduce nivel: ";
    cin >> num;
    if (num == 5)
        cout << "Muy alto" << endl;
    else if (num == 4)
        cout << "Alto" << endl;
    else if (num == 3)
        cout << "Medio" << endl;
    else if (num == 2)
        cout << "Bajo" << endl;
    else if (num == 1)
        cout << "Muy bajo" << endl;
    else
        cout << "Valor no válido" << endl;
    return 0;
}
```

Si num == 5 entonces Muy alto
Si num == 4 entonces Alto
Si num == 3 entonces Medio
Si num == 2 entonces Bajo
Si num == 1 entonces Muy bajo

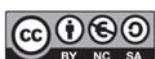


¿Código repetido en las distintas ramas?

```
if (num == 5) cout << "Muy alto" << endl;
else if (num == 4) cout << "Alto" << endl;
else if (num == 3) cout << "Medio" << endl;
else if (num == 2) cout << "Bajo" << endl;
else if (num == 1) cout << "Muy bajo" << endl;
else cout << "Valor no válido" << endl;
```



```
if (num == 5) cout << "Muy alto";
else if (num == 4) cout << "Alto";
else if (num == 3) cout << "Medio";
else if (num == 2) cout << "Bajo";
else if (num == 1) cout << "Muy bajo";
else cout << "Valor no válido";
cout << endl;
```



Selección múltiple Switch



La instrucción switch

Selección entre valores posibles de una expresión

Si `num == 5` → Muy alto

Si `num == 4` → Alto

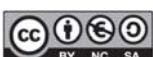
Si `num == 3` → Medio

Si `num == 2` → Bajo

Si `num == 1` → Muy bajo

```
switch (expresión) {  
    case exp1: instrucciones1  
    case exp2: instrucciones2  
    ...  
    case expN: instruccionesN  
    default: instruccionesD  
}
```

```
switch (num) {  
    case 5: cout << "Muy alto"; break;  
    case 4: cout << "Alto"; break;  
    case 3: cout << "Medio"; break;  
    case 2: cout << "Bajo"; break;  
    case 1: cout << "Muy bajo"; break;  
    default: cout << "Valor no válido";  
}
```



La instrucción break

Interrumpe el switch; continúa en la instrucción que siga

```
switch (num) {  
    case 5: cout << "Muy alto"; break;  
    case 4: cout << "Alto"; break;  
    case ③: cout << "Medio"; break;  
    case 2: cout << "Bajo"; break;  
    case 1: cout << "Muy bajo"; break;  
    default: cout << "Valor no válido";  
}
```

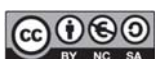


```
switch (num) {  
    case 5: cout << "Muy alto";  
    case 4: cout << "Alto";  
    case ③: cout << "Medio";  
    case 2: cout << "Bajo";  
    case 1: cout << "Muy bajo";  
    default: cout << "Valor no válido";  
}
```



Un menú

```
int main(){  
    int opcion = menu(4);  
    while (opcion != 0) {  
        switch (opcion) {  
            case 1: ... break;  
            case 2: ... break;  
            case 3: ... break;  
            case 4: ... break;  
            default: {}  
        }  
        opcion = menu(4);  
    }  
    ...  
}
```



Casos múltiples

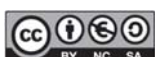
```
void mostrarNota(int nota) {
    switch (nota) {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4: cout << "Suspendido"; break; // De 0 a 4: SS
        case 5:
        case 6: cout << "Aprobado"; break; // 5 o 6: AP
        case 7:
        case 8: cout << "Notable"; break; // 7 u 8: NT
        case 9:
        case 10: cout << "Sobresaliente"; break; // 9 o 10: SB
        default: cout << "¡Valor no válido!";
    }
}
```



Un menú

```
int menu(){
    int op;
    cout << "1 - Nuevo registro" << endl;
    cout << "2 - Editar registro" << endl;
    cout << "3 - Eliminar registro" << endl;
    cout << "4 - Ver registro" << endl;
    cout << "0 - Salir" << endl;

    cout << "Opción: ";
    cin >> op;
    while((op < 0) || (op > 4)){
        cout << "¡Opción no válida! " << endl;
        cout << "Opción: ";
        cin >> op;
    }
    return op;
}
```



El menú con su bucle...

```
int main(){
    ...
    int opcion;
    opcion = menu();
    while (opcion != 0) {
        if (opcion == 1) {...}
        else if (opcion == 2) {...}
        else if (opcion == 3) {...}
        else /*if (opcion == 4)*/ {...}

        opcion = menu();
    }
    ...
}
```



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
```



Tipos de bucles

✓ Número de iteraciones condicionado :

– Bucle while

`while (condición) cuerpo`

Ejecuta el *cuerpo* mientras que la *condición* sea true.

Si inicialmente *condición* es false, *cuerpo* no se ejecuta.

– Bucle do-while

`do { cuerpo } while (condición);`

Ejecuta el *cuerpo* al menos una vez.

✓ Número de iteraciones conocido:

– Bucle for

`for (inicialización; condición; paso) cuerpo`

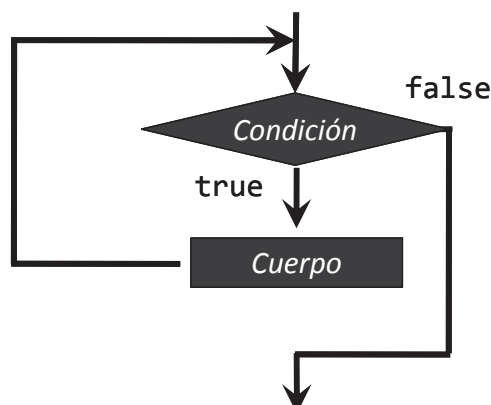
Ejecuta el *cuerpo* mientras que la *condición* sea true.



El bucle while

Mientras la condición sea cierta, ejecutar el cuerpo

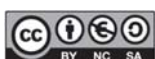
`while (condición) cuerpo`



cuerpo: instrucción simple o bloque

¿condición false al empezar?

No se ejecuta el *cuerpo* ninguna vez



Suma y media de números

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

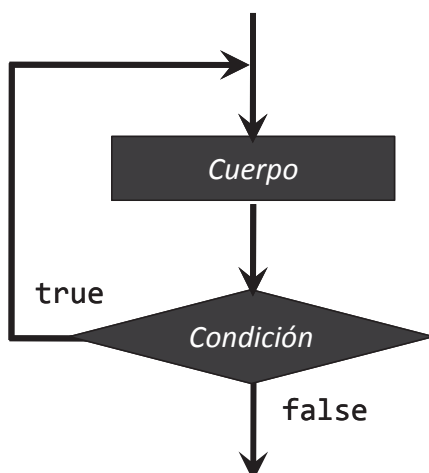
← Leemos el siguiente



El bucle do-while

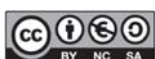
Número de iteraciones condicionado

do *cuerpo* while (*condición*); Condición al final del bucle



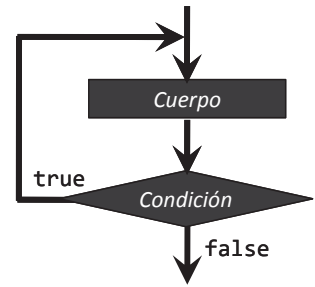
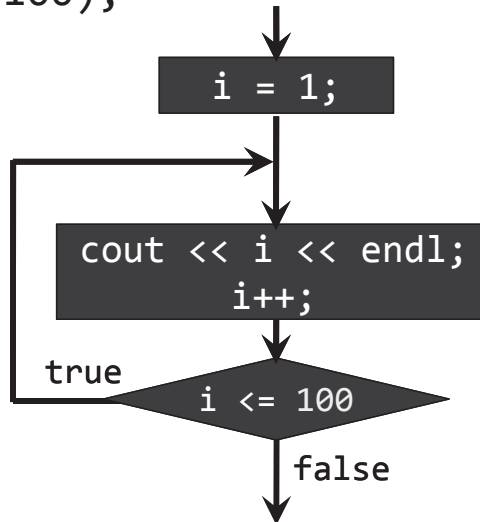
cuerpo: instrucción simple o bloque

El *cuerpo* siempre se ejecuta al menos una vez



Ejecución del bucle do-while

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



El cuerpo
se ejecuta
al menos
una vez



Ejemplo de bucle do-while

```
int menu(int max){
    int op;
    mostrarMenu();
    do {
        cout << "Opción: ";
        cin >> op;
        if ((op < 0) || (op > max))
            cout << "¡Opción no válida!";
    } while ((op < 0) || (op > max));

    return op;
}
```



¿Cuál elegir?

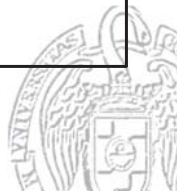
¿Ha de ejecutarse al menos una vez el cuerpo del bucle?

```
cin >> d; // Lectura del 1º
while (d != 0) {
    prod = prod * d;
    cont++;
    cin >> d; // Lectura
}
```

```
do {
    cin >> d; // Lectura
    if (d != 0) { // ¿Final?
        prod = prod * d;
        cont++;
    }
} while (d != 0); // ¿Final?
```

```
cout << "Opción: ";
cin >> op; // Lectura del 1º
while ((op < 0) || (op > 4)) {
    cout << "Error. Opción: ";
    cin >> op; // Lectura
}
```

```
do {
    cout << "Opción: "; // Error?
    cin >> op; // Lectura
} while ((op < 0) || (op > 4));
```



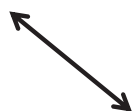
Bucle for

Número de iteraciones conocido

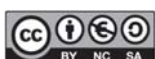
Variable contadora que determina el número de iteraciones

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

1, 2, 3, 4, 5, ..., 100

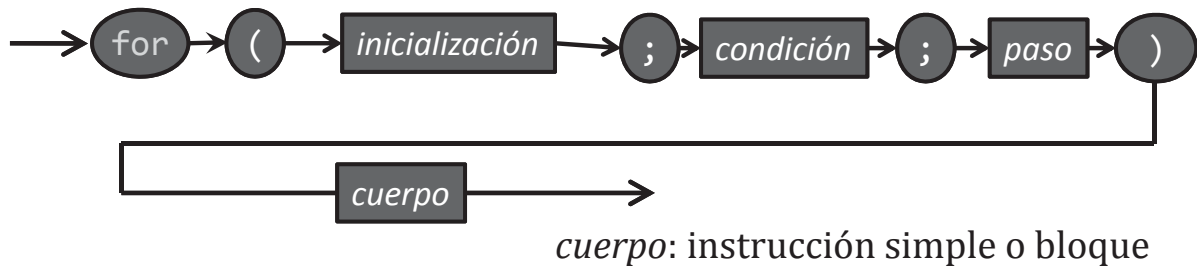


```
for (int i = 1; i <= 100; i++)
    cout << i << endl;
```



Bucle for

`for (inicialización; condición; paso) cuerpo`



Equivalente a

```
inicialización;
while (condición) {
    cuerpo
    paso
}
```



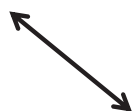
Bucle for

Número de iteraciones conocido

Variable contadora que determina el número de iteraciones

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

100, 99, 98, 97, ..., 2, 1



```
for (int i = 100; i >= 1; i--)
    cout << i << endl;
```



Bucle for

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



Bucle for

Bucles infinitos

```
for (int i = 1; i != 100; i = i + 2) ...
```

1 3 5 ... 99 101 103 105...

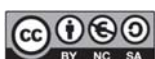
Cada vez más lejos del valor final

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



Ejemplo de bucle for

Número de iteraciones conocido al ejecutarse el bucle

```
#include <iostream>
using namespace std;
void cuentaAtras(int num){
    cout << "Cuenta atrás:" << endl;
    for (int i = num; i >= 0; i--){
        cout << i << endl;
    }

    int main() {
        int num;
        cout << "Número inicial: ";
        cin >> num;
        if (num < 0)
            cout << "¡El número debe ser positivo!" << endl;
        else cuentaAtras(num);
        return 0;
    }
```

Muestra la *secuencia* de números
num, num -1, num -2, ..., 2, 1, 0



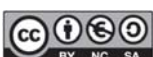
Bucles for anidados

Un bucle for en el cuerpo de otro bucle for

```
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
...	



Bucles while anidados

Un bucle while en el cuerpo de otro bucle while

```
int i = 1;
while (i <= 100) {
    int j = 1; // ERROR COMUN: No inicializar la variable
    while (j <= 5) {
        cuerpo
        j++;
    }
    i++;
}
```

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

...



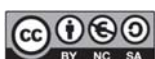
Ejemplo de bucles for anidados

Tablas de multiplicación

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

void mostrarTablas(int from, int to) {
    for (int i = from; i <= to; i++)
        for (int j = 1; j <= 10; j++)
            cout << setw(2) << i << " x "
                << setw(2) << j << " = "
                << setw(3) << i * j << endl;
}
```

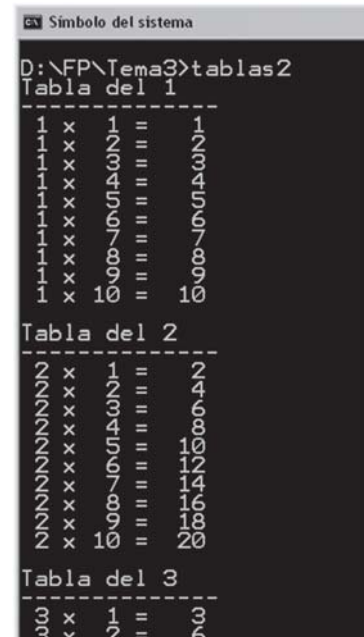
```
int main() {
    mostrarTablas(1,10); ...
}
```



Mejor presentación

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

void mostrarTablas(int from, int to) {
    for (int i = from; i <= to; 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;
    }
}
```



Fundamentos de la programación

Ámbito y visibilidad



Ámbito de los identificadores

Cada bloque (o bucle for) crea un nuevo ámbito:

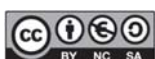
```
int main() {  
    double d, suma = 0;  
    int cont = 0;  
    cin >> d;  
    while (d != 0)  
    {  
        if (d > 0)  
        {  
            suma += d;  
            cont++;  
        }  
    }  
    cout << "Suma = " << suma << endl;  
    cout << "Media = " << suma / cont << endl;  
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

3 ámbitos anidados



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 anidados



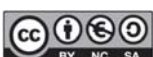
Ámbito de los identificadores

```
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 la variable i



Ámbito de la variable contadora

Declarada en el propio bucle

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

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

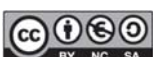
No se puede usar en instrucciones que sigan al bucle

Declarada antes del bucle

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

Se conoce en el cuerpo del bucle y después del mismo

Ámbito externo al bucle



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;
}
```

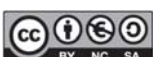
Ámbito de la variable c



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



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;  
}
```



Entrada/Salida con archivos de texto



Archivos de texto y archivos binarios

Archivo de texto: secuencia de caracteres (código binario ASCII)

T	o	t	a	:		1	2	3	.	4	↵	A	...	<EOF>
---	---	---	---	---	--	---	---	---	---	---	---	---	-----	-------

Archivo binario: contiene una secuencia de códigos binarios

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

(Códigos representados en notación hexadecimal)

Secuencias de *bytes* terminada en EOF (End Of File)

Los archivos se manejan en los programas por medio de *flujos de bytes (stream)*.

Archivo de texto: *flujo de caracteres organizados en líneas mediante el uso de una marca : <EOL> (↵)*

Similar a la E/S por consola



Flujos de entrada y de salida para archivos

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

- ✓ Lectura del archivo, flujo de entrada, variables de tipo `ifstream`
- ✓ Escritura en el archivo, flujo de salida, variables de tipo `ofstream`

Biblioteca `fstream`

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



Lectura de archivos de texto

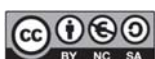
Flujos de texto de entrada

`ifstream`

Para leer de un archivo de texto:

- 1 Declara una variable de tipo `ifstream`
- 2 Asocia la variable con el archivo de texto (*abre el archivo*)
- 3 Realiza las lecturas por medio del operador `>>` (extractor), y las funciones `get(char)` y `getline(istream, string)`
- 4 Desliga la variable del archivo de texto (*cierra el archivo*)

Extractor (`>>`): igual que en la E/S por consola, primero se saltan los *separadores* (espacios, tab, intro, ...)

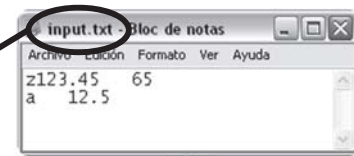


Flujos de texto de entrada

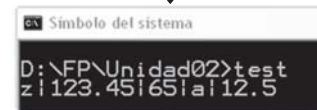
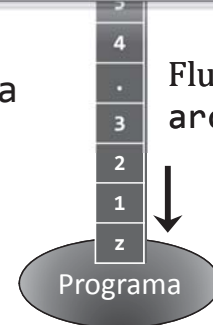
```
int i;  
char c1, c2;  
double d1, d2;
```

- 1 `ifstream archivo;`
- 2 `archivo.open("input.txt"); // Apertura`
- 3 `archivo >> c1 >> d1 >> i >> c2 >> d2;`
- 4 `archivo.close(); // Cierre`

```
cout << c1 << "|" << d1 << "|" << i  
     << "|" << c2 << "|" << d2;
```



Flujo de entrada
archivo



Flujos de texto de entrada

Apertura del archivo

Conecta la variable con el archivo de texto del dispositivo

```
flujo.open(cadena);
```

```
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

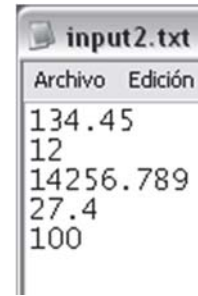
```
flujo.close();  
archivo.close();
```



Ejemplo de lectura de un archivo

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

int main() {
    double d;
    ifstream archivo;
    archivo.open("input2.txt"); // Apertura
    if (!archivo.is_open())
        cout << "¡No se pudo abrir el archivo!" << endl;
    else {
        archivo >> d;
        cout << d << endl;
        archivo >> d;
        cout << d << endl;
        archivo.close(); // Cierre del archivo
    }
    return 0;
}
```



Si el archivo tiene
menos de 2 números,
la lectura fallará



Flujos de texto de entrada

Final de archivo: eof()

Si durante la ejecución de una operación de lectura se alcanza la marca <EOF>, la variable del archivo queda marcada al estado `eof` cierto.

Para preguntar por este estado se dispone de la función `eof()`

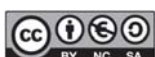
`flujo.eof()`: `true` si se ha leído la marca <EOF>
`false` en caso contrario

Fallo de lectura: fail()

Si durante la ejecución de una operación de lectura se produce un error de formato, la variable del archivo queda marcada al estado `fail` cierto.

Para preguntar por este estado se dispone de la función `fail()`

`flujo.fail()`: `true` si se produjo un fallo
`false` en caso contrario



Ejemplo lectura de un archivo de texto

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
// suponiendo que el texto del archivo son num. enteros
// num0 num1 ... numi ... numN <EOF>
int sumArch(string nombre){
    ifstream datos; int num; int total = 0;
    datos.open(nombre); // Apertura
    if(datos.is_open()){
        datos >> num; // Lectura del primero
        // sumados todos los números anteriores al último leído
        while(!datos.fail()){ // mientras lectura sin fallo
            total = total + num; // Proceso
            datos >> num; // Lectura del siguiente
        }
    }
    // else
    datos.close(); // Cierre del archivo

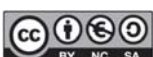
    return total;
}
```



Ejemplo lectura de un archivo de texto

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int sumArch(string nombre);

int main(){
    int suma = 0;
    suma = sumArch("datosnum.txt");
    cout << "Suma del archivo: " << suma;
    cout << "Nombre del archivo: ";
    string nombre;
    cin >> nombre;
    suma = sumArch(nombre);
    cout << "Suma del archivo: " << suma;
    return 0;
}
```



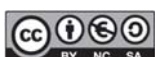
Ejemplo lectura de un archivo de texto

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
// suponiendo que el texto del archivo son num. enteros
// num0 num1 ... numi ... numN <EOF>
bool buscar(string nombre, int buscado){
    ifstream datos; int num; bool encontrado;
    datos.open(nombre); // Apertura
    if(datos.is_open()){
        datos >> num ; //leer primero
        // buscar no está entre los anteriores al último leído
        // mientras lectura sin fallo y no encontrado
        while(!datos.fail() && num != buscado)
            datos >> num; // leer siguiente
        encontrado = !datos.fail(); // -> num == buscar
    }
    // else
    datos.close(); // Cierre del archivo
    return encontrado;
}
```



Ejemplo lectura de un archivo de texto

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
bool buscar(string nombre, int buscado);
int main(){
    string nombre; int datoBuscar;
    cout << "Nombre del archivo: ";
    cin >> nombre;
    cout << "Dato a buscar: ";
    cin >> datoBuscar;
    if(buscar(nombre, datoBuscar))
        cout << " Dato encontrado " << endl;
    else cout << " Dato NO encontrado " << endl;
    return 0;
}
```



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!

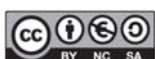
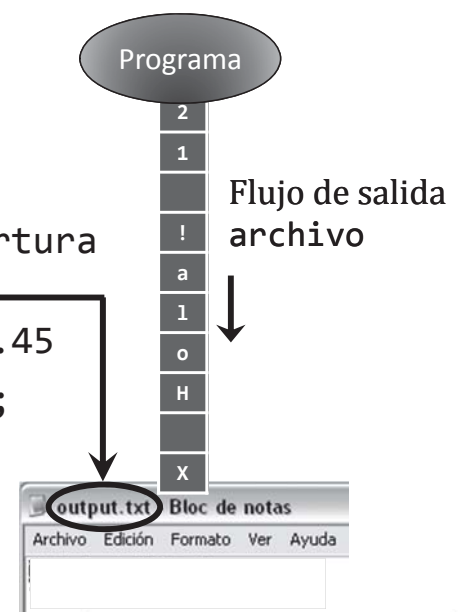
Si no se cierra el archivo se puede perder información



Flujos de texto de salida

```
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

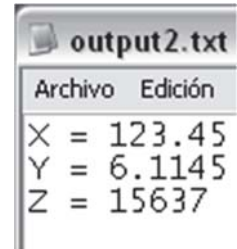


Ejemplo de escritura en un archivo

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

int main()
{
    ofstream archivo;
    archivo.open("output2.txt"); // Apertura
    archivo << "X = " << 123.45 << endl;
    archivo << "Y = " << 6.1145 << endl;
    archivo << "Z = " << 15637 << endl;
    archivo.close(); // Cierre del archivo

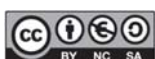
    return 0;
}
```



Ejemplo copia de un archivo de texto

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

bool copiar(string origen, string destino){
    ofstream salida; ifstream entrada; char ch;
    salida.open(destino); // Apertura
    entrada.open(origen); // Apertura
    if(entrada.is_open() && salida.is_open()){
        entrada.get(ch); // Lectura
        // copiados todos los anteriores al último leído
        while(!entrada.eof()) { // Mientras no final de archivo
            salida.put(ch); // Escritura: put(char)
            entrada.get(ch); // Lectura
        }
        salida.close(); // Cierre del archivo
        entrada.close(); // Cierre del archivo
        return true;
    }
    else return false;
}
```



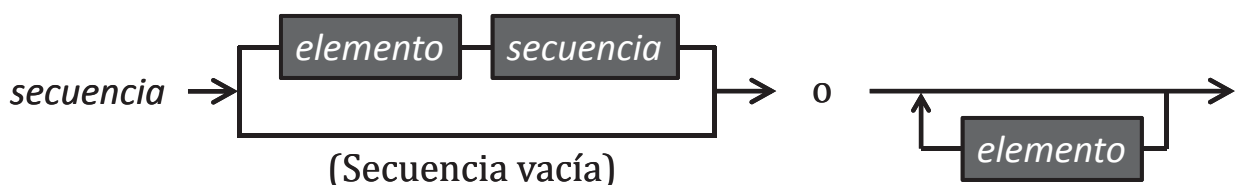
Secuencias



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

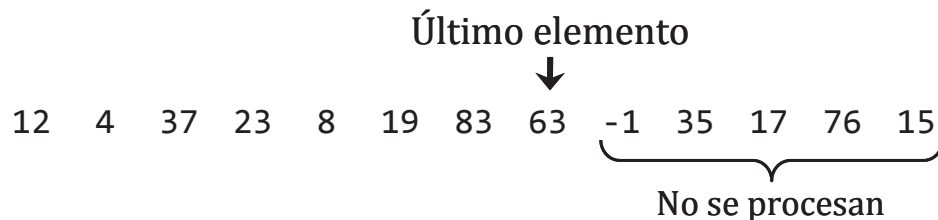
Cadenas de caracteres:

¿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:

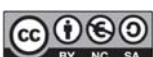
Obtener el siguiente elemento

Procesar el elemento

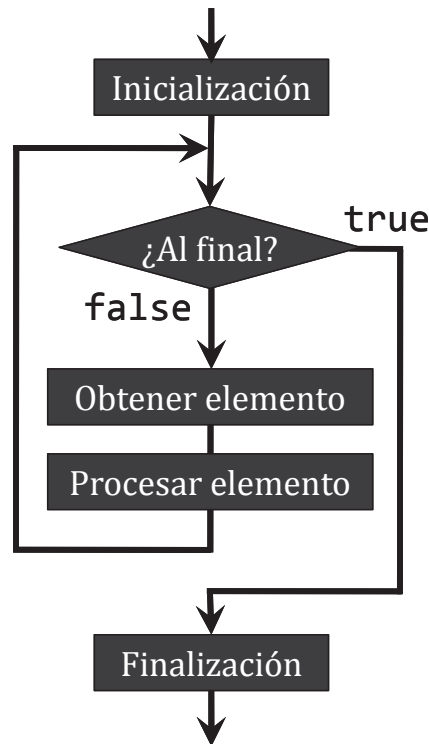
Finalización

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

Inicialización

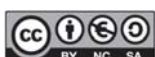
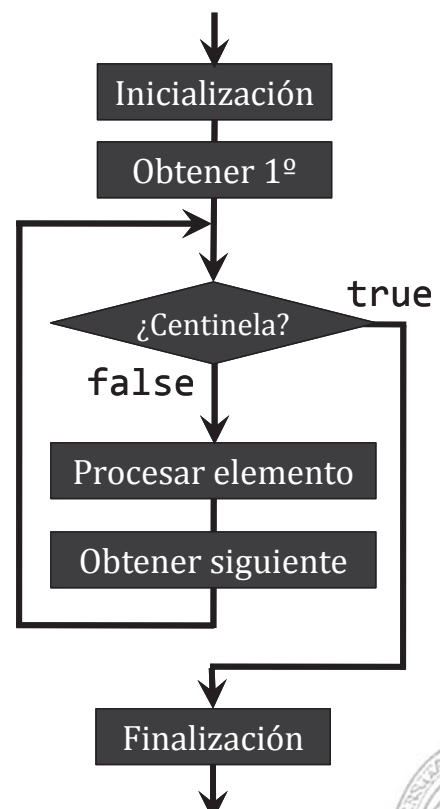
Obtener el primer elemento

Mientras no sea el centinela:

Procesar el elemento

Obtener el siguiente elemento

Finalización



Secuencias explícitas con centinela

Implementación con do-while

Inicialización

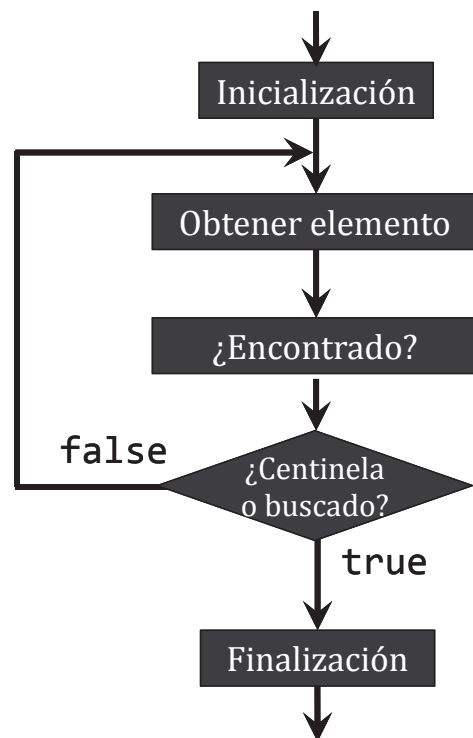
Hacer:

Obtener el siguiente elemento

¿Encontrado?

Mientras ni encontrado ni el centinela

Finalización (¿encontrado?)



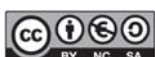
Secuencias explícitas leídas del teclado

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; } Siguiendo 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



Secuencias explícitas leídas del teclado

Primer número mayor que uno dado

do-while

Centinela: -1 (no debe cumplir la propiedad buscada)

```
double d, num;
cout << "Encontrar primero mayor que: ";
cin >> num;
bool encontrado;
do {
    cout << "Siguiendo (-1 para terminar): ";
    cin >> d;
    encontrado = (d > num);
} while ((d != -1) && !encontrado);

if (encontrado)
    cout << "Encontrado";
else
    cout << "No encontrado";
```

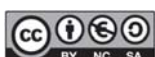


Secuencias explícitas leídas de archivo

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;
}
```



Secuencias calculadas



Secuencias calculadas

sumatorio.cpp

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

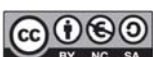
1 2 3 4 5 6 7 8 ...

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

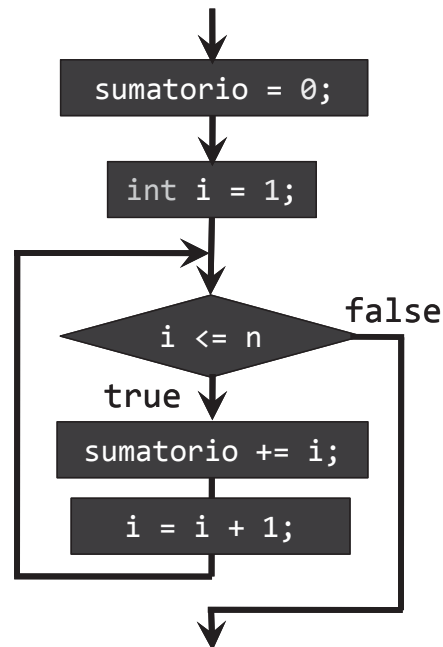
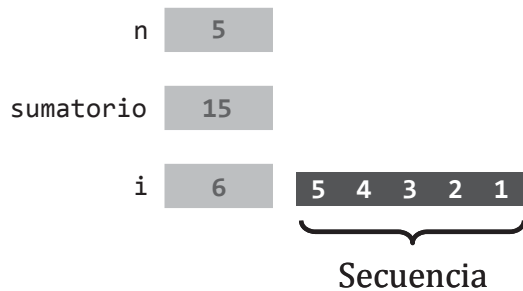
Último elemento de la secuencia: n



Suma de una secuencia calculada

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

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



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



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
}
    fib 5
    fibMenos1 3
    fibMenos2 2
```

0 1 1 2 3 5 ...



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

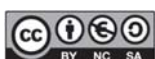
Finalización

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

Elemento que se busca: satisfará una condición

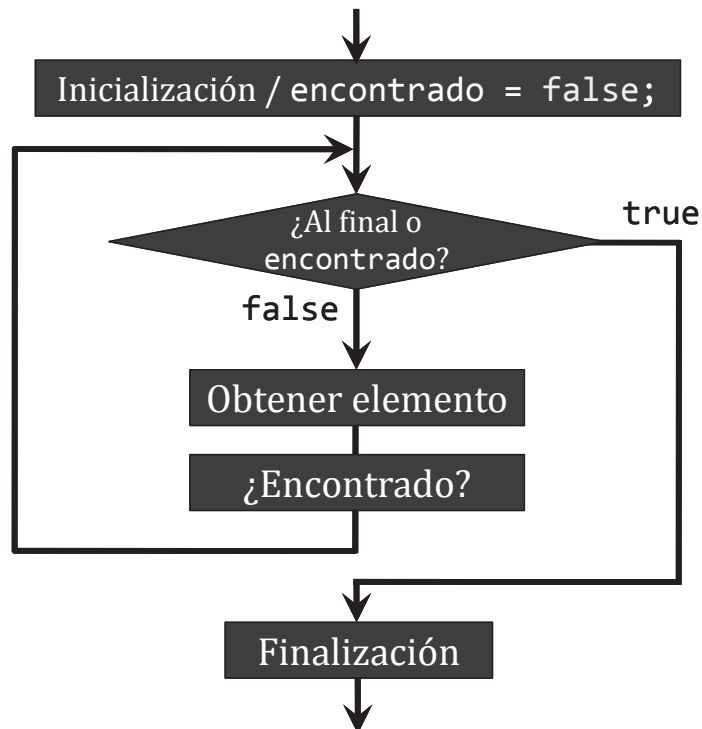
Dos condiciones de terminación del bucle: se encuentra / al final

Variable lógica que indique si se ha encontrado



Esquema de búsqueda

Localización del primer elemento con una propiedad



Secuencias explícitas con centinela

Implementación con while

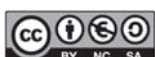
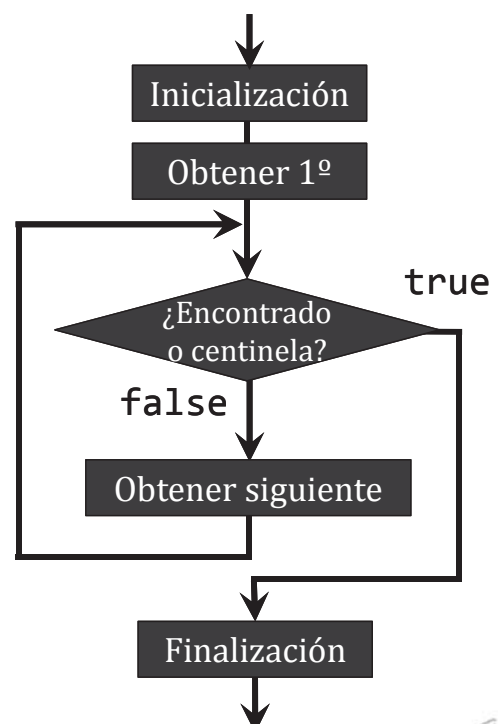
Inicialización

Obtener el primer elemento

Mientras ni encontrado ni el centinela:

Obtener el siguiente elemento

Finalización (¿encontrado?)



Secuencias explícitas leídas del teclado

Primer número mayor que uno dado

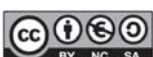
busca.cpp

```
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
    }
}
```



Fundamentos de la programación

Tipos de datos



Variables

Memoria suficiente para su tipo de datos.

`sizeof(tipo);` devuelve el número de bytes utilizado para una variable de ese tipo

```
short int i = 3;
```

```
int j = 9;
```

```
char c = 'a';
```

```
double x = 1.5;
```

j 9

i 3

c a

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

- ❖ Primitivos: `int`, `float`, `double`, `char`, `bool`
(y sus posibles modificadores: `unsigned`, `long`, `short`)
Conjunto de valores predeterminado
- ❖ Definidos en el programa: *enumerados*
Conjunto de valores definido por el programador

✓ Estructurados

- ❖ Colecciones homogéneas: *arrays*
Todos los elementos de la colección de un mismo tipo
- ❖ Colecciones heterogéneas: *estructuras* (Tema 5)
Elementos de la colección de tipos distintos.



Conversiones automáticas de tipos

Promoción de tipos

Dos operandos de tipos distintos pero compatibles:

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

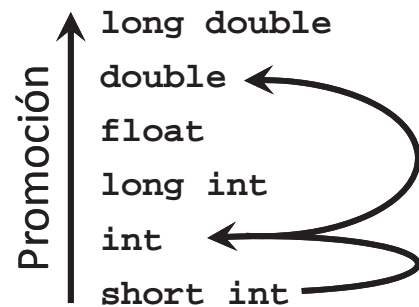
```
short int i = 3;
int j = 2;
double a = 1.5, b;
b = a + i * j;
```

```
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*)

Fuerza a que el valor resultante de la *expresión* se trate como un valor de ese *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



Declaración de tipos

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

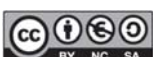
```
typedef unsigned short int tPos;
```

```
tPos positivo = 0;
```



descripción

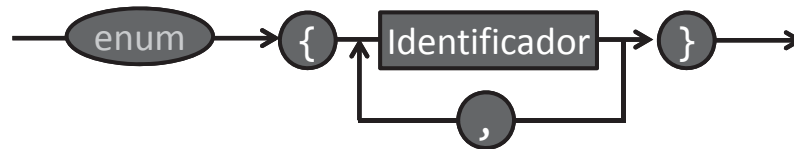
Declaración de tipo frente a definición de variable



Tipos enumerados

Enumeración del conjunto de valores posibles para variables

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



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

Valores literales (en amarillo) que pueden tomar las variables

Constantes de tipo entero, que comienzan por defecto en 0, y secuencialmente se incrementa en 1, terminando en (número de elementos enumerados - 1).
(centimo es 0, dos_centimos es 1, cinco_centimos es 2, ..., euro es 6)



Tipos enumerados

Mejoran la legibilidad

```
typedef descripción nombre_de_tipo;
```

Elegimos un nombre para el tipo: tMoneda

```
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 sus valores

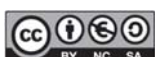
```
moneda1 = dos_centimos;
```

```
moneda2 = euro;
```

moneda1 dos_centimos

moneda2 euro

(Internamente se usan enteros)



Entrada/salida para tipos enumerados

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

Lectura de la variable ddls:

```
cin >> ddls;
```

Se espera un valor entero. Si el usuario escribe directamente lunes o jueves, no se puede leer.

Y si se escribe la variable en la pantalla:

```
cout << ddls;
```

Se verá un número entero

→ Código de entrada/salida específico



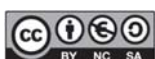
Lectura del valor de un tipo enumerado

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

```
tDiaSemana leerDiaSemana() {  
    int i;  
    cout << "Dia de la semana (1..7): ";  
    cin >> i;  
    return tDiaSemana(i - 1);  
}
```

Molde para obtener
el valor del tipo
(Los enteros equivalentes
empiezan en 0)

```
tDiaSemana ddls = leerDiaSemana();
```



Escritura de variables de tipos enumerados

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

```
void mostrarDiaSemana(tDiaSemana ddls) {
    if (ddls == lunes)      cout << "lunes";
    else if (ddls == martes) cout << "martes";
    else if (mes == miercoles) cout << "miércoles";
    else if (mes == jueves)  cout << "jueves";
    else if (mes == viernes) cout << "viernes";
    else if (mes == sabado)   cout << "sábado";
    else /* (mes == domingo) */ cout << "domingo";
}
```

```
tDiaSemana ddls = leerDiaSemana();
mostrarDiaSemana(ddls);
```



Escritura de variables de tipos enumerados

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

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



Tipos enumerados

Son tipos ordenados (posición en la enumeración)

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

```
const int NumDias = 7;
```

```
tDiaSemana ddls;
```

```
...
```

```
if (ddls == jueves)...
```

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

No admiten operadores de incremento y decremento

Podemos definir la función `incr()` utilizando moldes

```
ddls = incr(ddls); // No se puede: ddls = ddls+1; ddls++; ...
```

```
tDiaSemana incr(tDiaSemana dia){  
    return tDiaSemana((int(dia) + 1) % NumDias);  
}
```

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

Definir constante para el cardinal de los enumerados



Ejemplo de tipos enumerados

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

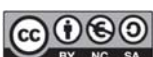
También se pueden
definir tipos locales

```
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;
```

```
int main()  
{  
    tDiaSemana ddls;  
    int dia;  
    tMes mes;  
    int anio;  
    ...
```

```
// Mostramos la fecha  
cout << "Hoy es: ";  
mostrarDia(ddls);  
cout << " " << dia << " de ";  
mostrarMes(mes);  
cout << " de " << anio <<  
endl;  
...  
}
```



Arrays de tipos simples



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



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 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;
```

```
typedef double tTemp[7];  
typedef short int tDiasMes[12];  
typedef char tVocales[5];  
typedef double tVentas[31];
```

tempMax

?	?	?	?	?	?	?
0	1	2	3	4	5	6

```
tDiasMes diasMes;
```

diasMes

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

```
tVocales vocales;
```

vocales

?	?	?	?	?
0	1	2	3	4

```
tVentas ventasFeb;
```

ventasFeb

?	?	?	?	?	?	?	?	?	?	?	?	?	...	?
0	1	2	3	4	5	6	7	8	9	10	11	12		30



NO se inicializan automáticamente



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]

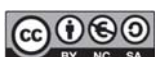
Procesamiento de cada elemento:

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]? ¿O ventas[-1]? ¿O ventas[132]?

¡Memoria de alguna otra variable del programa!



Define los tamaños de los arrays con constantes



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];  
  
double tempMedia(const tTemp temp) {  
    double total = 0;  
    for (int i = 0; i < Dias; i++) // Recorrido del array  
        total = total + temp[i];  
  
    return total / Dias;  
}
```



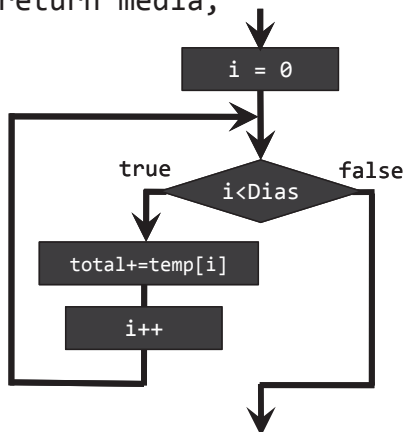
Los arrays , como datos de entrada para las funciones,
se pasan como constantes



Recorrido de arrays

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

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



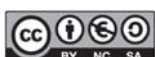
	Memoria
Dias	7
temp[0]	12.40
temp[1]	10.96
temp[2]	8.43
temp[3]	11.65
temp[4]	13.70
temp[5]	13.41
temp[6]	14.07
media	?
total	84.62
i	7



Recorrido de arrays

```
#include <iostream>  
using namespace std;  
  
const int Dias = 7;  
typedef double tTemp[Dias];  
  
double tempMedia(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: " << tempMedia(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



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;
}
```



Capacidad de los arrays

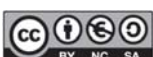
La capacidad de un array no puede ser alterada en la ejecución

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 con asignación
(aunque sean del mismo tipo) :

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

Han de copiarse los elementos uno a uno:

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



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

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

✓ Podemos usar un contador de elementos...

```
tArray lista;  
int contador = 0;
```

contador: indica cuántas posiciones del array se utilizan

Sólo accederemos a las posiciones entre 0 y contador-1

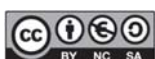
Las demás posiciones no contienen información relevante

✓ Podemos usar un centinela...

```
const double Marca = ...;
```

Sólo accederemos a las posiciones entre 0 y k, para k la primera posición tal que `(lista[k+1] == Marca)`

Hay que dejar hueco para marca



Arrays no completos

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

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 ((contador < Max) && (valor != -1)) {
            lista[contador] = valor;
            contador++;
            archivo >> valor;
        } ...
    }
```



Arrays no completos

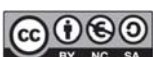
```
        archivo.close();
        med = media(lista, contador);
        cout << "Media de los elementos de la lista: " << med << endl;
    }
    else {
        cout << "¡No se pudo abrir el archivo!" << endl;
    }

    return 0;
}

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

Sólo recorreremos hasta cont-1

¡Cuidado!



Arrays no completos

```
#include <iostream> ...

const int Max = 100; const double Marca = -1;
typedef double tArray[Max+1];

double media(const tArray lista);

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



Arrays no completos

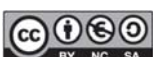
```
        archivo.close();
        med = media(lista);
        cout << "Media de los elementos de la lista: " << med << endl;
    }
    else {
        cout << "¡No se pudo abrir el archivo!" << endl;
    }

    return 0;
}

double media(const tArray lista) {
    double total = 0; int ind = 0;
    while (lista[ind] != Marca) {
        total = total + lista[ind];
        ind++;
    }
    return total / ind;
}
```

Sólo recorreremos hasta Marca

¡Cuidado!



Enumerados como índices

```
const int Cuantas = 15; const int NumMonedas = 7;
typedef enum { centimo, dos_centimos, cinco_centimos,
    diez_centimos, veinte_centimos, medio_euro, euro } tMoneda;
typedef int tCalderilla[NumMonedas];
string aCadena(tMoneda moneda);
// Devuelve la cadena correspondiente al valor de moneda

tCalderilla bolsillo; // Monedas de cada tipo
bolsillo[centimo] = 1;
bolsillo[dos_centimos] = 2;
bolsillo[cinco_centimos] = 2;
bolsillo[diez_centimos] = 1;
bolsillo[veinte_centimos] = 3;
bolsillo[medio_euro] = 3;
bolsillo[euro] = 2;
for (int moneda= centimo; moneda < NumMonedas; moneda= incr(moneda))
    cout << aCadena(moneda) << " " << bolsillo[moneda] << endl;
```



Fundamentos de la programación

Cadenas de caracteres



Cadenas de caracteres

Arrays de caracteres

Cadenas: secuencias de caracteres de longitud variable

"Hola" "Adiós" "Supercalifragilístico" "1234567"

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

- ✓ Arrays de tipo `char` con una longitud máxima
- ✓ Un último carácter especial al final: `'\0'`

Tipo `string`

- ✓ Cadenas más sofisticadas (contienen la longitud de la cadena como dato asociado)
- ✓ Sin longitud máxima (gestión automática de la memoria)
- ✓ Multitud de funciones de utilidad



Cadenas de caracteres al estilo de C

Arrays de caracteres terminado en el carácter nulo

```
typedef char tCadena[15];
```

```
tCadena cadena = "Adiós"; // Inicialización al declarar
```

Al inicializar o leer un array de caracteres se coloca al final el carácter nulo (código ASCII 0 – `'\0'`)

Indica que en esa posición termina la cadena (exclusive)

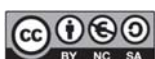
cadena	A	d	i	ó	s	\0									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

En el array caben *dimensión*-1 caracteres significativos

Longitud máxima de la variable `cadena`: 14

No se pueden asignar cadenas literales: ~~`cadena = "Hola";`~~

(Ni copiar cadenas directamente: ~~`cad2 = cad1;`~~)



Cadenas de caracteres al estilo de C

Entrada/salida por consola

```
tCadena cadena;  
cin >> cadena; // Se añade un nulo al final  
cout << cadena << endl; // El nulo no se muestra
```

Extractor: la lectura termina en el primer separador

No se comprueba si se leen más caracteres de los que caben:
¡Riesgo de sobrescribir otras zonas de memoria!

setw(): máximo de caracteres a colocar (incluyendo el nulo)

```
cin >> setw(15) >> cadena;
```

Función getline(*cadena_estilo_C*, *max*):

También lee espacios en blanco

```
cin.getline(cadena, 15); // Hasta 14 caracteres o '\n'
```



Cadenas de caracteres al estilo de C

Funciones (biblioteca cstring)

- ✓ strlen(*cadena*): longitud actual de la *cadena*

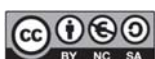
```
cout << "Longitud: " << strlen(cadena);
```
- ✓ strcpy(*destino*, *origen*): copia de *cadena origen* en *cadena destino*

```
strcpy(cad2, cad1);          strcpy(cad, "Me gusta C++");
```
- ✓ strcat(*destino*, *origen*):
añade (*concatena*) una copia de *origen* al final de *destino*

```
typedef char tCad[80];  
tCad cad1 = "Hola", cad2 = "Adiós";  
strcat(cad1, cad2); // Ahora cad1 contiene "HolaAdiós"
```
- ✓ strcmp(*cad1*, *cad2*): compara las cadenas y devuelve 0 si son iguales, un positivo si *cad1* > *cad2* o un negativo si *cad1* < *cad2*
Compara lexicográficamente (*alfabéticamente*)

```
tCad cad1 = "Hola", cad2 = "Adiós";  
strcmp(cad1, cad2); // Un positivo ("Hola" > "Adiós")
```

<http://www.cplusplus.com/reference/cstring/>



Ejemplo de cadenas al estilo de C

```
#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");
    ...
}
```

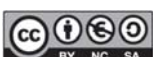


Ejemplo de cadenas al estilo de C

```
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;
}
```

```
Símbolo del sistema
D:\FP\Tema5>cadenas
Me gusta C++
Cadena: me gusta más Java
me
Cadena: me gusta más Java
me gusta más Java
Longitud: 17
Hola amigo
Hola amigo es mayor que amigo
D:\FP\Tema5>
```



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;
```






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.

