

3

Tipos e instrucciones II

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

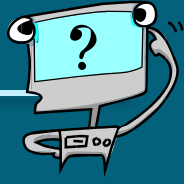
Material de la Prof.^a Mercedes Gómez Albarrán
Versión revisada y ampliada del material del Prof. Luis Hernández Yáñez

Facultad de Informática
Universidad Complutense



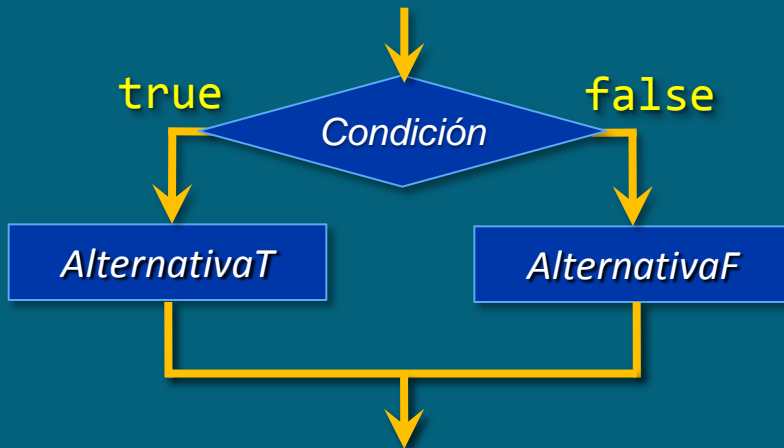
Más condicionales y bucles

Flujo de ejecución

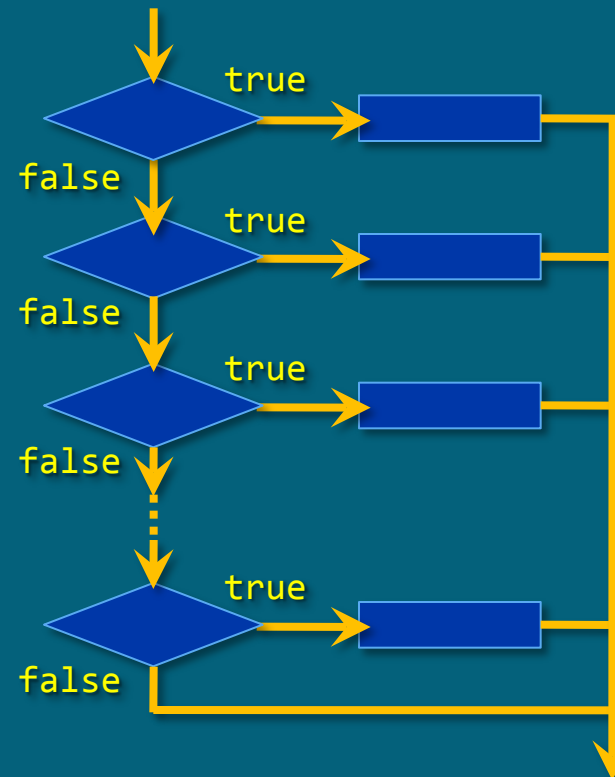


Selección

2 caminos: Selección simple



> 2 caminos: Selección múltiple



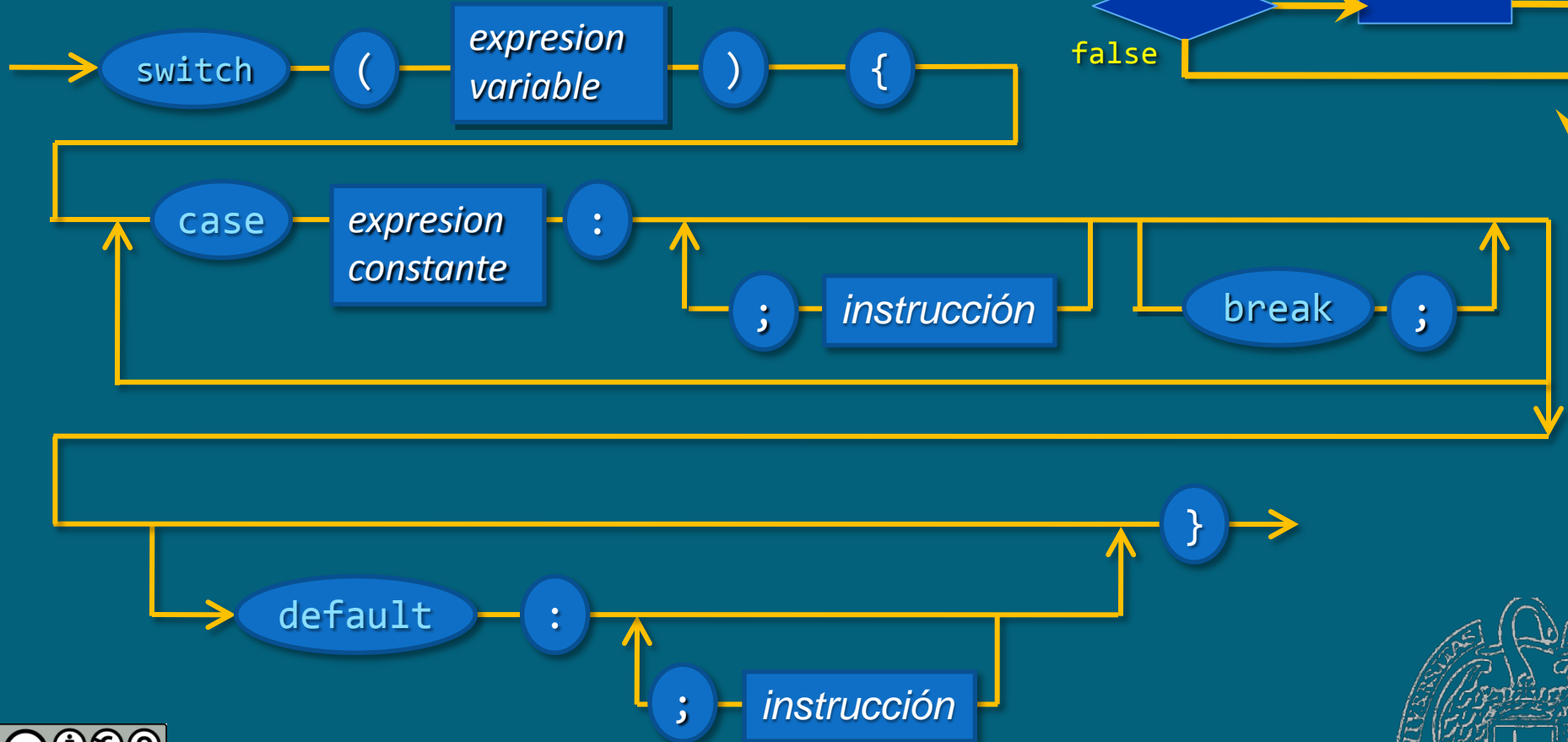
Diagramas de flujo

(no confundir con diagramas sintácticos)



Selección múltiple: la instrucción `switch`

sentencia `switch`



Selección múltiple: la instrucción `switch`

nivel2.cpp

Ejemplo.- Nivel de un valor

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

```
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";  
}
```



Selección múltiple: la instrucción `switch`

La instrucción `break`

Interrumpe el `switch`, llevando la ejecución al final del mismo.

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



```
Num: 3  
Medio
```

```
Num: 3  
Medio  
Bajo  
Muy bajo  
Valor no válido
```



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



Selección múltiple: la instrucción `switch`

```
switch (num * otra) {    // se admiten expresiones

case (5 * 1) + 0: cout << "Muy alto"; break;
// se admiten expresiones "constantes"

/* case 4 * otra: cout << "Alto"; break; */
// no se permiten expresiones que tengan variables

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";
}
```



Selección múltiple: la instrucción `switch`

Ejemplo.- Un uso común de `switch`: el tratamiento de menús

```
cout << "1 - Suma" << endl;
cout << "2 - Resta" << endl;
cout << "3 - Producto" << endl;
cout << "4 - Division" << endl;
cout << "0 - Salir" << endl;
cout << "Opcion: " << endl;
cin >> num;
switch (num) {
    case 0: break;
    case 1: /* instrucciones suma */ break;
    case 2: /* instrucciones resta */ break;
    case 3: /* instrucciones producto */ break;
    case 4: /* instrucciones division */ break;
    default: cout << "¡Valor no válido!";
}
```

Algoritmo:

- Mostrar el menú
- Solicitar y leer la opción
- Si opcion = suma ...

sino si opcion = resta ...

sino



Selección múltiple: la instrucción `switch`

nota2.cpp

Ejemplo.- Nota simbólica equivalente a numérica

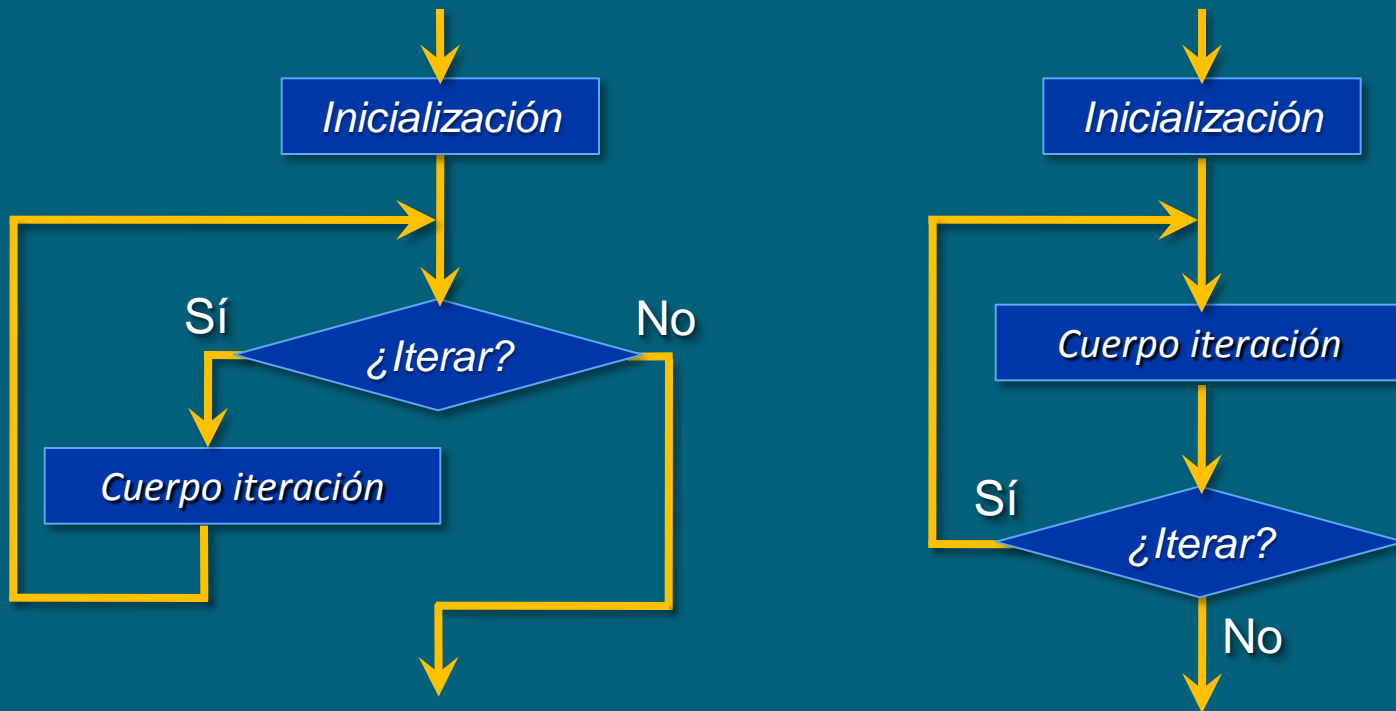
```
int num;
cout << "Nota: " << endl;
cin >> num;
switch (num) {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4: cout << "Suspenso"; break; // De 0 a 4: Suspenso
    case 5:
    case 6: cout << "Aprobado"; break; // 5 o 6: Aprobado
    case 7:
    case 8: cout << "Notable"; break; // 7 u 8: Notable
    case 9:
    case 10: cout << "Sobresaliente"; break; // 9 o 10: Sobresaliente
    default: cout << "¡Valor no válido!";
}
```



Flujo de ejecución



Repetición o iteración



Diagramas de flujo

(no confundir con diagramas sintácticos)



Repetición

Tipos de bucles

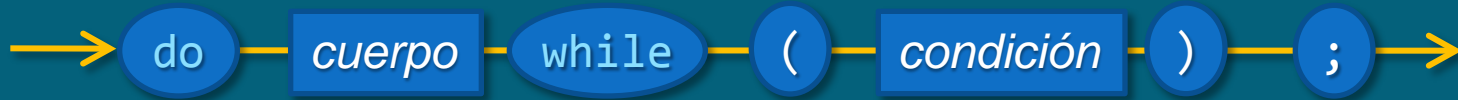
	Comprobación pre-iteración	Comprobación post-iteración
Recorrido fijo	<code>for</code> <code>while</code>	<code>do-while</code>
Recorrido variable	<code>while</code>	<code>do-while</code>



Repetición: bucle `do-while`

Recorrido fijo + comprobación post-iteración

Recorrido variable + comprobación post-iteración



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

El *cuerpo* siempre se ejecuta al menos una vez. La condición se comprueba tras cada pasada.

El *cuerpo* del bucle es una instrucción (simple o compuesta –bloque –).



Repetición: bucle `do-while`

Ejemplo.- Bucle `do-while` usado para recorrido fijo (equivalente a

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

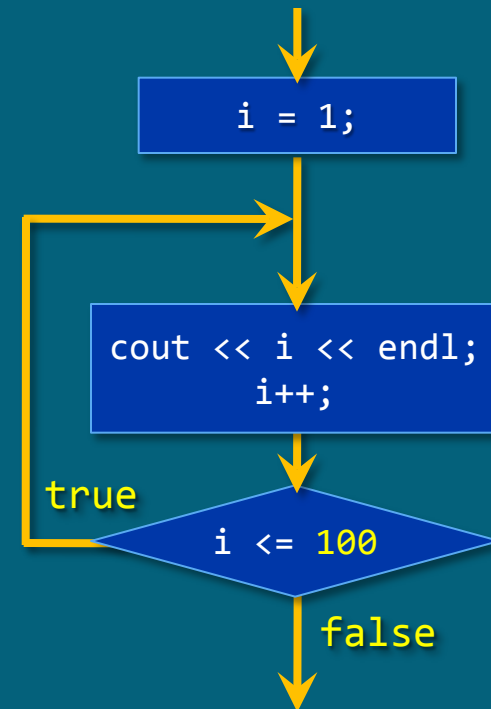
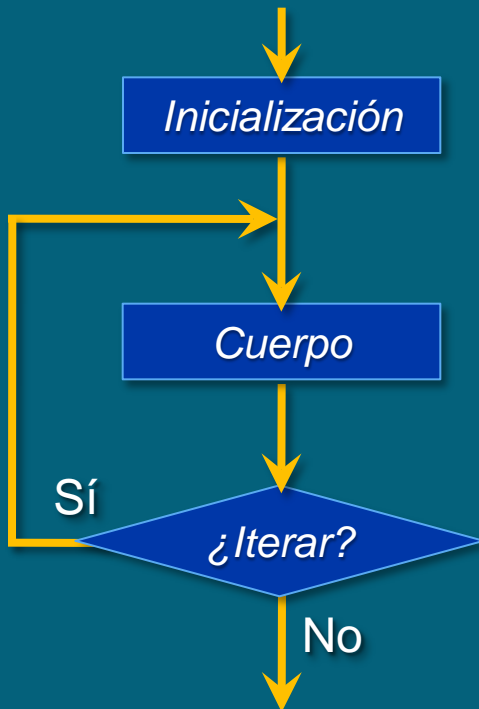
```
int i = 1;
do {
    cout << i << endl;
    i++;
} while (i <= 100);
```

`do.cpp`



Repetición: bucle do-while

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



Repetición: bucle `do-while`

Ejemplo.- Mejorando el programa del área del triángulo mediante el uso del bucle `do-while`.

Algoritmo4:

- Repetir solicitar y leer los datos (base y altura) mientras que los datos leídos no sean válidos
- Calcular el área
- Visualizar el área

...

```
do {
    cout << "Introduzca la base del triangulo: "; cin >> base;
    cout << "Introduzca la altura del triangulo: "; cin >> altura;
} while ( altura <= 0 || base <= 0 )
area = base * altura / 2;
cout << "El area de un triangulo es: " << area << endl;
```



Repetición: bucle `do-while`

Ejemplo.- Número de dígitos que tiene un número entero.

¿Algoritmo?

¿Programa?



Repetición: bucle do-while

Ejemplo.- Control en menús

```
do {
    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 >> num;
} while (num < 0 || num > 4);
switch (num) {
    case 0: break;
    case 1: /* codigo de la opcion 1 */ break;
    case 2: /* codigo de la opcion 2 */ break;
    case 3: /* codigo de la opcion 3 */ break;
    case 4: /* codigo de la opcion 4 */ break;
    default: cout << "¡Valor no válido!";
}
```



Implementa una función que haga una recogida controlada de la opción del menú



Ámbito y visibilidad

Ámbito y visibilidad

Cada bloque crea un nuevo ámbito dentro del ámbito en que se encuentra.

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

    return 0;
}
```

3 ámbitos anidados



Ámbito y visibilidad

Ámbito de los identificadores: Un identificador se conoce (se puede usar) en el ámbito en el que está declarado, a partir de su instrucción de declaración, y en los subámbitos posteriores a su declaración.

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



Ámbito y visibilidad

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).

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



Archivos de texto

Entrada / salida con archivos de texto

Archivos

La memoria de la computadora es volátil.

Medios (dispositivos) de almacenamiento permanente:

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

...



La información se guarda en esos dispositivos en forma de *archivos*.

Un archivo es una secuencia de datos que constituyen la información de un determinado documento.



Entrada / salida con archivos de texto

Archivos de texto y binarios

- Archivo de texto: contiene una 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.)

Lo que signifiquen los códigos dependerá del programa que haga uso del archivo.

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

Los archivos de texto se leen y escriben desde un programa en C++ por medio de flujos de texto.



Entrada / salida con archivos de texto

Flujos de texto

Los flujos de texto `cin` y `cout` que se usan para la E/S por consola se crean automáticamente al comenzar la ejecución del programa, asociándose a los dispositivos teclado y pantalla, respectivamente.

Los flujos de texto asociados a archivos de texto se han de crear explícitamente durante la ejecución del programa, asociando cada uno al archivo de texto correspondiente.

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

- Flujos de entrada (lectura): objetos de tipo `ifstream`.
- Flujos de salida (escritura) : objetos de tipo `ofstream`.

Los tipos `ifstream` y `ofstream` están declarados en la biblioteca `fstream`.



Entrada con archivos de texto

Flujos de texto de entrada

ifstream

Para leer de un archivo de texto:

- 1 Se declara una variable flujo de tipo **ifstream**
- 2 Se asocia la variable con el archivo de texto (*se abre el archivo*)
`flujo.open(nombre_del_archivo_con_extension);`
- 3 Se realizan las lecturas por medio del operador >> (extractor)
Mismo comportamiento que en la E/S por consola
- 4 Se desliga la variable del archivo de texto (*se cierra el archivo*)
Se invoca la función `close()` sobre la variable de tipo **ifstream**:

`flujo.close();`



¡Atención!

Si no existe el archivo con el que se intenta asociar el flujo, se produce un error de ejecución.



Entrada con archivos de texto

Ejemplo de lectura de datos desde un archivo de texto:

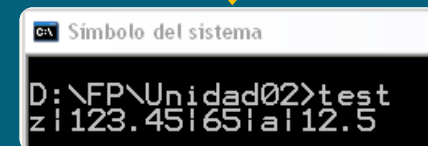
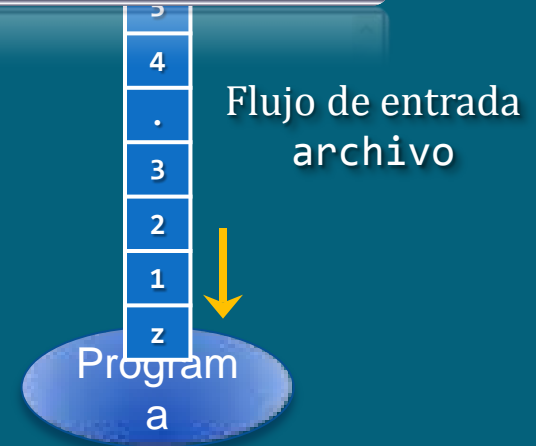
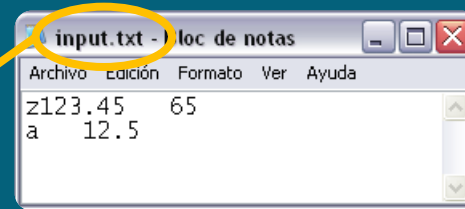
```
int i;
```

```
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 del archivo`

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



Salida con archivos de texto

Flujos de texto de salida

ofstream

Para crear un archivo de texto y escribir en él información necesitamos un flujo de texto de salida (variable de tipo **ofstream**).

Para escribir en un archivo de texto:

- 1 Se declara una variable de tipo **ofstream**
- 2 Se asocia la variable con el archivo de texto (*se abre el archivo*)
- 3 Se realizan las escrituras por medio del operador << (insertor)
- 4 Se desliga la variable del archivo de texto (*se cierra el archivo*)



¡Atención!

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



¡Atención!

Si no se cierra el archivo se puede perder algo de la información escrita.

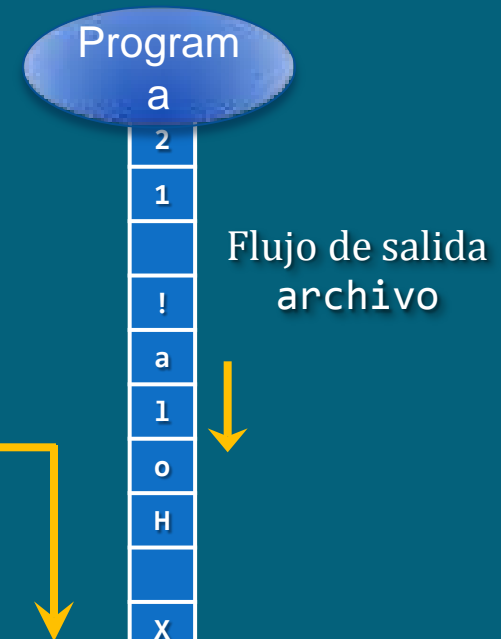


Salida con archivos de texto

Ejemplo de escritura de datos en un archivo 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 del archivo`



Otras funciones de flujos de texto

Flujos de archivos de texto

Otras funciones definidas:

- `is_open()`: Función booleana que devuelve **true** si el archivo se ha abierto correctamente y **false** si se ha producido algún error.

```
ifstream archivo;  
archivo.open("input.txt");  
cout << "Abierto: " << boolalpha << archivo.is_open();
```

- `eof()`: Función booleana que devuelve **true** si se ha alcanzado el final del archivo y **false** en caso contrario. Es una función útil para flujos de entrada (`ifstream`), para no leer más allá del final.

```
char c;  
ifstream archivo;  
archivo.open("input.txt");  
archivo >> c;  
cout << "Al final: " << boolalpha << archivo.eof();
```



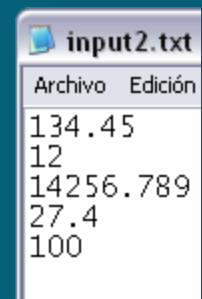
Ejemplo de entrada con archivos de texto

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

int main()
{
    double d;
    ifstream archivo;
    archivo.open("input2.txt"); // Apertura
    archivo >> d;
    cout << d << endl;
    archivo >> d;
    cout << d << endl;
    archivo >> d;
    cout << d << endl;
    archivo.close(); // Cierre del archivo

    return 0;
}
```

leer.cpp



Si el archivo tiene menos de 3 números,
la lectura en archivo fallará (error de ejecución).



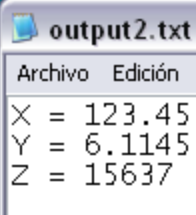
Ejemplo de salida con archivos de texto

```
#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;
}
```

escribir.cpp



Archivo	Edición
X =	123.45
Y =	6.1145
Z =	15637



Esquemas de recorrido y búsqueda

Secuencias

Secuencia = Sucesión de elementos de un mismo tipo.

Tipo de secuencia		Final
Explícita	Introducida por teclado	Longitud conocida de antemano
		¿Finalizar o seguir?
		Elemento centinela (valor especial que no es válido en la secuencia)
	En archivo	Marca fin de archivo o elemento centinela
	En memoria	Longitud de la lista o último elemento
Calculada / recurrente		Propiedad del último elemento a calcular



Esquemas de tratamiento de secuencias

Recorrido

- Desplazamiento por todos los elementos de la secuencia
- Termina cuando se alcanza el final de la secuencia (independientemente de si existe orden o no)

Búsqueda

- Desplazamiento que puede finalizar antes de pasar por todos los elementos de la secuencia
- Termina:
 - cuando se localiza el primer elemento que cumple la condición o cuando se alcanza el final de la secuencia (secuencia no ordenada)
 - cuando se localiza el primer elemento que cumple la condición, cuando se alcanza el final de la secuencia o cuando se pasa a explorar una zona de la secuencia donde no puede encontrarse lo buscado (secuencia ordenada)

Esquemas mixtos

- Combinan distintas búsquedas, o búsquedas y recorridos por subsecuencias de la secuencia original



Esquemas de tratamiento de secuencias

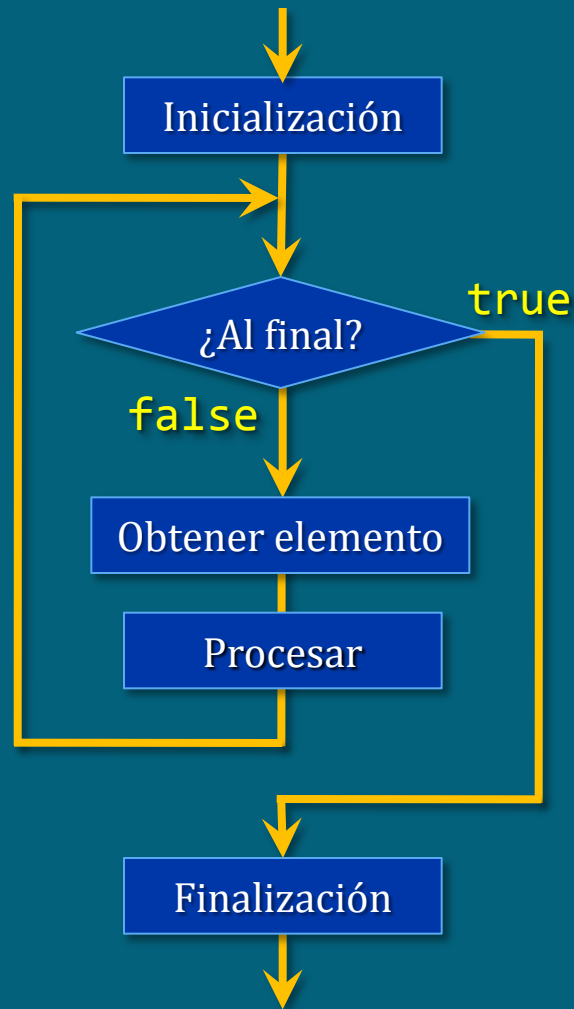
- Suma de una secuencia de números introducidos por teclado
- Suma de los números de un fichero
- Longitud de una cadena de caracteres leída de teclado
- Numero de caracteres que hay en una secuencia antes de la aparición de un cierto carácter
- Localización de un elemento en un fichero
- ¿Son iguales dos archivos?
- ¿Están los paréntesis de una cadena bien emparejados?
- Contabilizar el número de elementos iguales a uno dado en un archivo ordenado
- Escribir cadena introducida por teclado eliminando los blancos iniciales
- Generar copia de un archivo eliminando en todas las líneas los blancos iniciales.



¿Qué tipo de esquema es el apropiado?



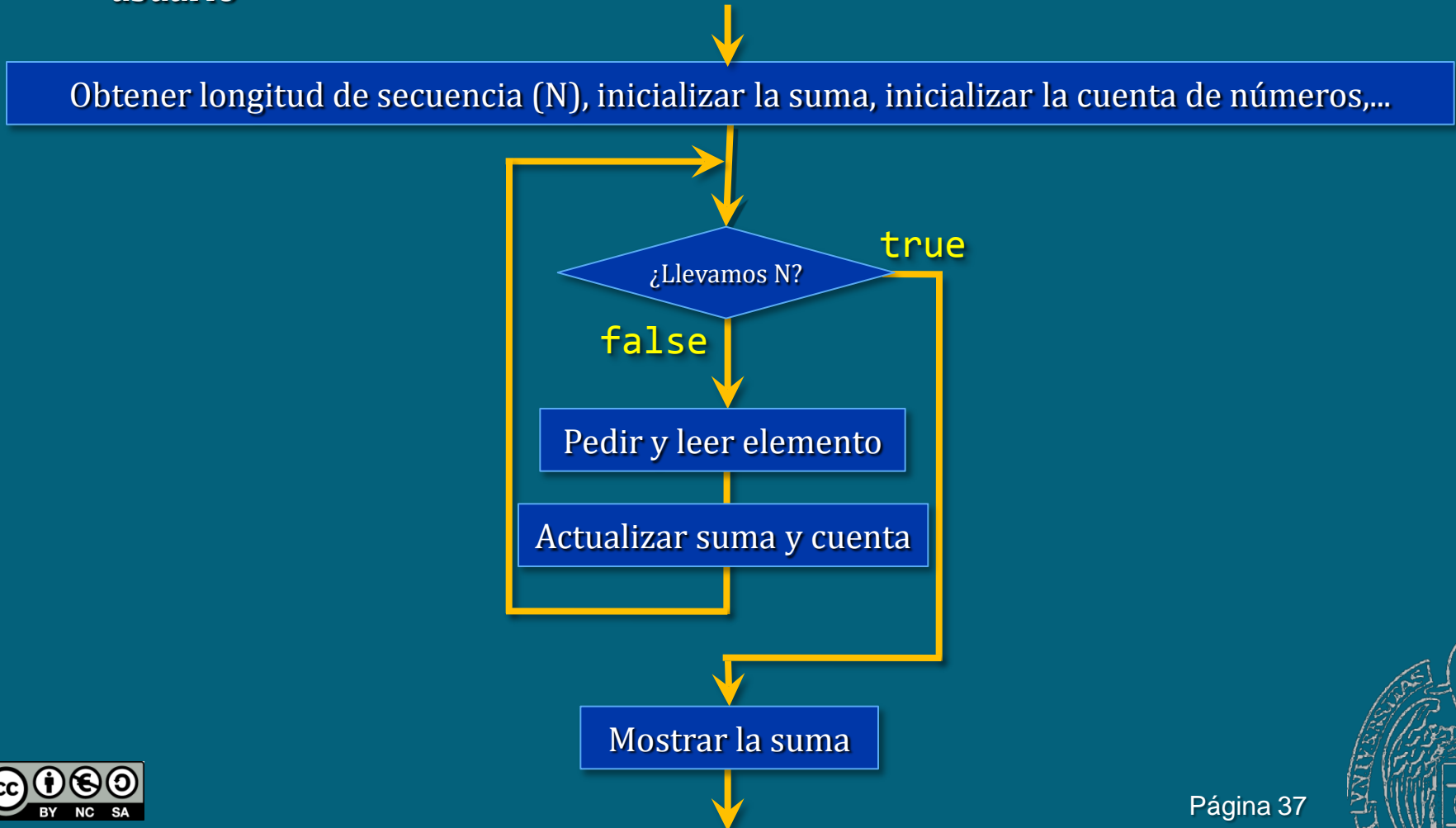
Esquema de recorrido



Esquema de recorrido

Ejemplo 1.- Secuencia explícita leída del teclado (longitud conocida)

Mostrar la suma de tantos números introducidos por teclado como haya indicado el usuario



Esquema de recorrido

Ejemplo 1.- Secuencia explícita leída del teclado (longitud conocida)

Mostrar la suma de tantos números introducidos por teclado como haya indicado el usuario

```
double dato, suma = 0;   int N;

cout << "Longitud de la secuencia: ";
cin >> N;
for (int i = 1; i <= N; i++) {
    cout << "Valor: "; cin >> dato;
    suma = suma + dato;
}
cout << "Suma = " << suma << endl;
```



Implementar la solución con
`while` y `do-while`



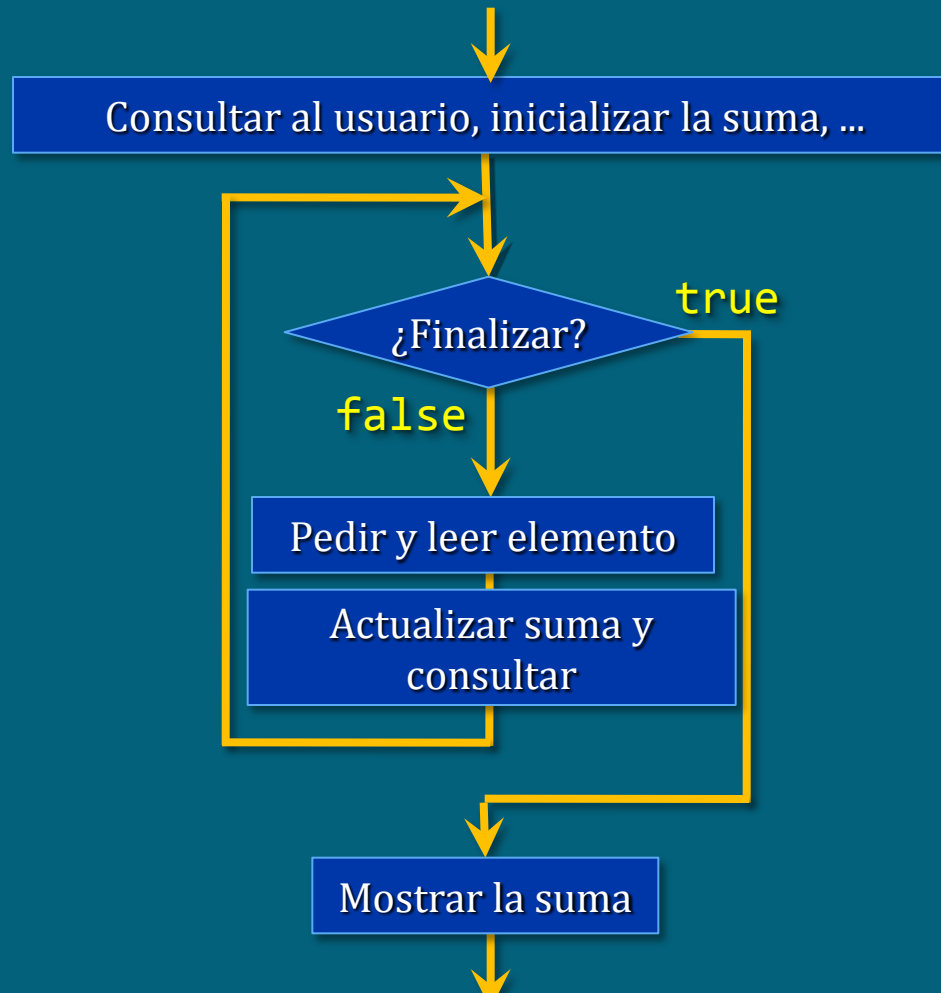
Implementar una función
que, dada la longitud de la
secuencia, devuelva la suma
de los elementos de la
misma



Esquema de recorrido

Ejemplo 2.- Secuencia explícita leída del teclado (¿finalizar?)

Mostrar la suma de números introducidos por teclado, consultando al usuario cuándo quiere dejar de introducir datos



Esquema de recorrido

Ejemplo 2.- Secuencia explícita leída del teclado (¿finalizar?)

Mostrar la suma de números introducidos por teclado, consultando al usuario cuándo quiere dejar de introducir datos

```
double suma = 0;    bool fin;    char c;
cout << "Terminar (S/N)? ";
cin >> c;
fin = toupper(c) == 'S';
while (!fin) {
    double dato;
    cout << "Valor: ";
    cin >> dato;
    suma = suma + dato;
    cout << "Terminar (S/N)? ";
    cin >> c;
    fin = toupper(c) == 'S';
}
cout << "Suma = " << suma << endl;
```

suma_finalizar_while.cpp



¿Qué hace el programa si se introduce una letra que no es ni S ni N?

Implementación
con *while*

Uso de indicador lógico con
semántica *finalizar*



Esquema de recorrido

Ejemplo 2.- Secuencia explícita leída del teclado (¿finalizar?)

Mostrar la suma de números introducidos por teclado, consultando al usuario cuándo quiere dejar de introducir datos

```
double suma = 0;    bool fin;
do {
    double dato;
    char c;
    cout << "Terminar (S/N)? ";
    cin >> c;
    fin = toupper(c) == 'S';
    cout << "Valor: ";
    cin >> dato;
    suma = suma + dato;
} while (!fin);
cout << "Suma = " << suma << endl;
```



¿Funciona de
manera apropiada?

Implementación
con **do-while**

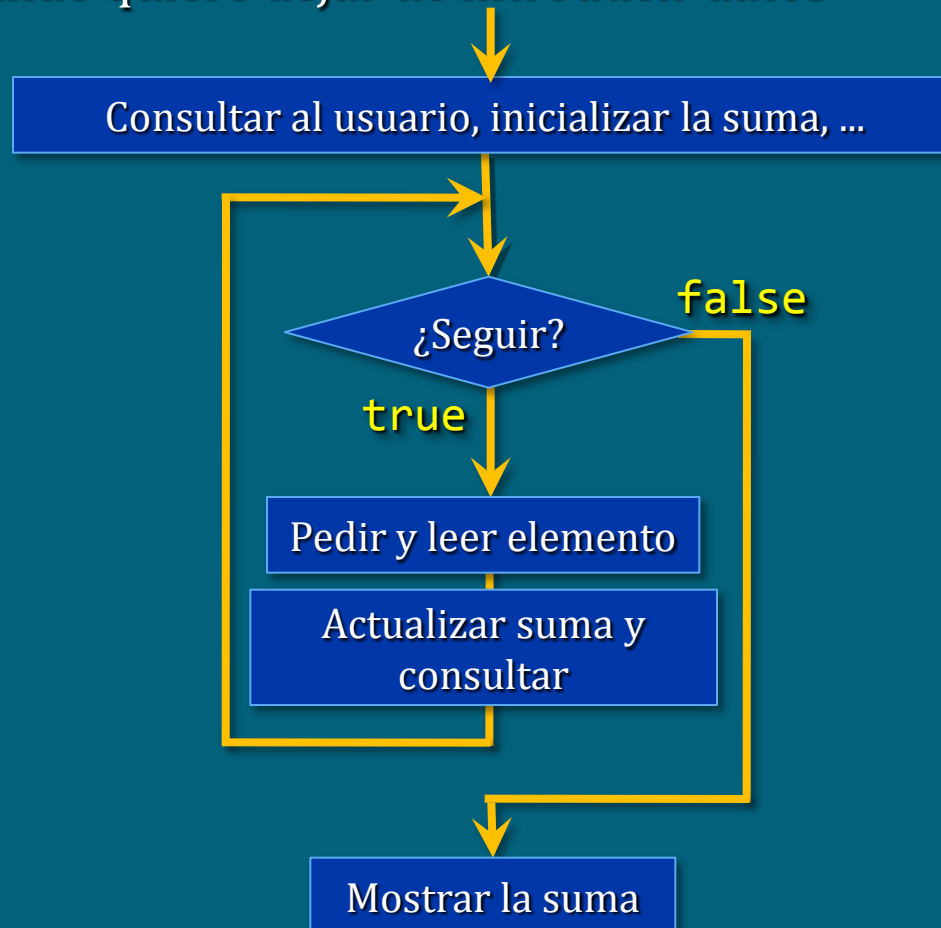
Uso de indicador lógico con
semántica *finalizar*



Esquema de recorrido

Ejemplo 2.- Secuencia explícita leída del teclado (¿seguir?)

Mostrar la suma de números introducidos por teclado, consultando al usuario cuándo quiere dejar de introducir datos



Esquema de recorrido

Ejemplo 2.- Secuencia explícita leída del teclado (¿seguir?)

Mostrar la suma de números introducidos por teclado, consultando al usuario cuándo quiere dejar de introducir datos

```
double suma = 0;  bool seguir;  char c;
cout << "Seguir (S/N)? ";
cin >> c;
seguir = toupper(c) == 'S';
while (seguir) {
    double dato;
    cout << "Valor: ";
    cin >> dato;
    suma = suma + dato;
    cout << "Seguir (S/N)? ";
    cin >> c;
    seguir = toupper(c) == 'S';
}
cout << "Suma = " << suma << endl;
```

Implementación
con `while`

Uso de indicador lógico con
semántica *seguir*



Esquema de recorrido

Ejemplo 2.- Secuencia explícita leída del teclado (¿seguir?)

Mostrar la suma de números introducidos por teclado, consultando al usuario cuándo quiere dejar de introducir datos

```
double suma = 0;    bool seguir;
do {
    double dato;    char c;
    cout << "Seguir (S/N)? ";
    cin >> c;
    seguir = toupper(c) == 'S';
    if (seguir) {
        cout << "Valor: ";
        cin >> dato;
        suma = suma + dato;
    }
} while (seguir);
cout << "Suma = " << suma << endl;
```

suma_seguir_do-while.cpp

Implementación
con `do-while`

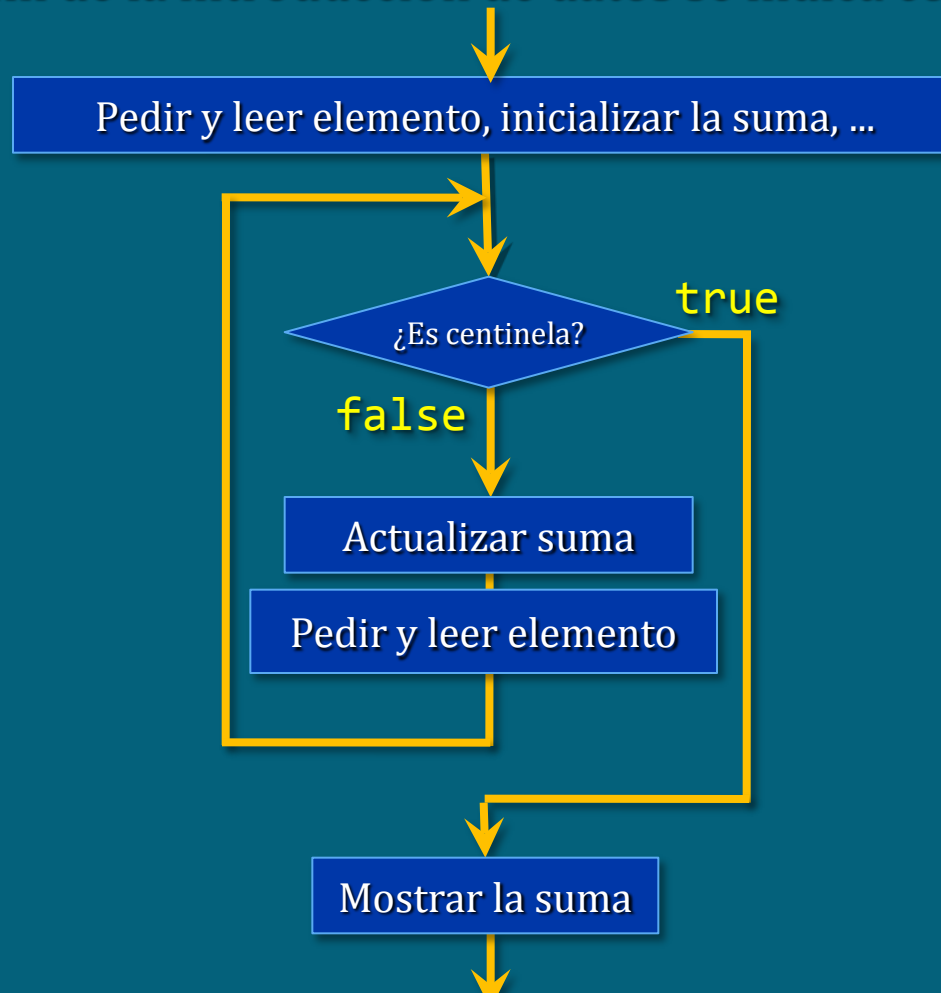
Uso de indicador lógico con
semántica *seguir*



Esquema de recorrido

Ejemplo 3.- Secuencia explícita leída del teclado (con centinela)

Mostrar la suma de los números reales que introduzca el usuario por teclado. El fin de la introducción de datos se indica con un 0.



Esquema de recorrido

Ejemplo 3.- Secuencia explícita leída del teclado (con centinela)

Mostrar la suma de los números reales que introduzca el usuario por teclado. El fin de la introducción de datos se indica con un 0.

```
double d, suma = 0;

cout << "Introduce un numero (0 para acabar): ";
cin >> d;
while(d != 0) {
    suma = suma + d;
    cout << "Introduce un numero (0 para acabar): ";
    cin >> d;
}
cout << "Suma = " << suma << endl;
```



Implementar la solución con
`do-while`



Esquema de recorrido

Ejemplo 3.- Secuencia explícita leída del teclado (con centinela)

Mostrar la suma de los números reales que introduzca el usuario por teclado. El fin de la introducción de datos se indica con un 0.

```
double d, suma = 0;
bool encontrado;

cout << "Introduce un numero (0 para acabar): ";
cin >> d;
encontrado = d == 0;
while(!encontrado) {
    suma = suma + d;
    cout << "Introduce un numero (0 para acabar): ";
    cin >> d;
    encontrado = d == 0;
}
cout << "Suma = " << suma << endl;
```

suma_centinela.cpp

Uso de indicador lógico con
semántica *encontrado*

Esquema de recorrido

Ejemplo 3.- Secuencia explícita leída del teclado (con centinela)

Mostrar la suma de los números reales que introduzca el usuario por teclado. El fin de la introducción de datos se indica con un 0.

```
double d, suma = 0;
bool no_encontrado;

cout << "Introduce un numero (0 para acabar): ";
cin >> d;
no_encontrado = .....;
while (.....) {
    suma = suma + d;
    cout << "Introduce un numero (0 para acabar): ";
    cin >> d;
    no_encontrado = .....;
}
cout << "Suma = " << suma << endl;
```



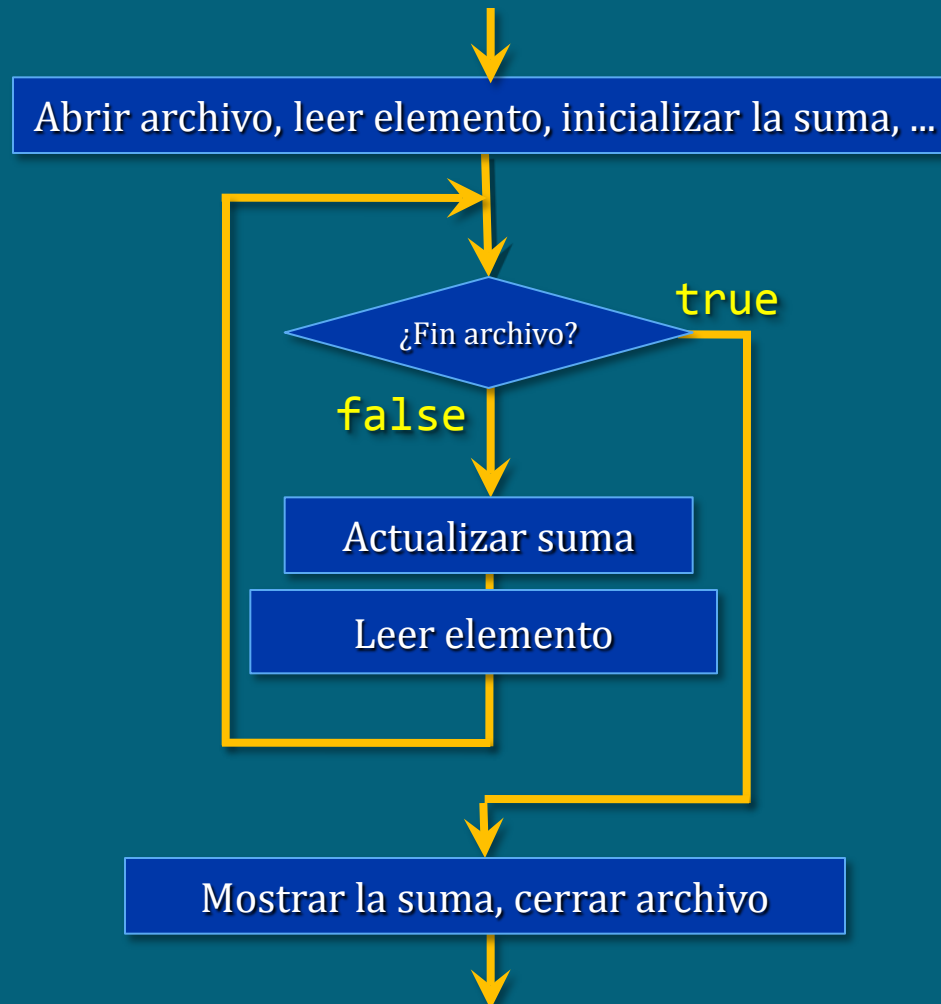
Completa los huecos

Uso de indicador lógico con
semántica *no_encontrado*

Esquema de recorrido

Ejemplo 4.- Secuencia explícita leída de archivo

Mostrar la suma de los números reales que hay en el archivo *input.txt*.



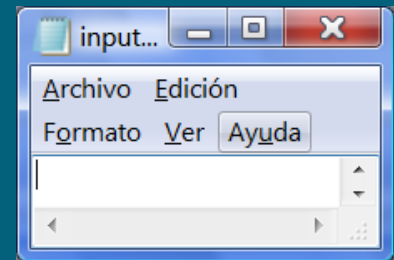
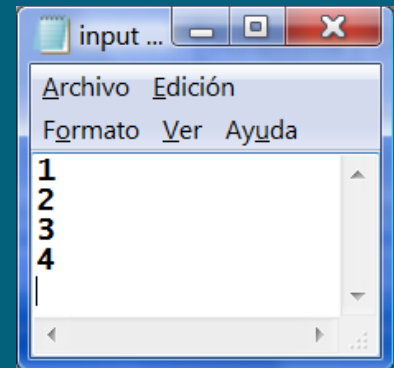
Esquema de recorrido

Ejemplo 4.- Secuencia explícita leída de archivo

Mostrar la suma de los números reales que hay en el archivo *input.txt*.

```
double d, suma = 0;    ifstream archivo;

archivo.open("input.txt");
if (archivo.is_open()) {
    archivo >> d;
    while (!archivo.eof()) {
        suma = suma + d;
        archivo >> d;
    }
    cout << "Suma = " << suma << endl;
    archivo.close();
}
```



Los ejemplos sobre archivos de texto que usan eof() suponen que los archivos están bien formateados (estructurados en líneas)

Esquema de recorrido

Ejemplo 4.- Secuencia explícita leída de archivo

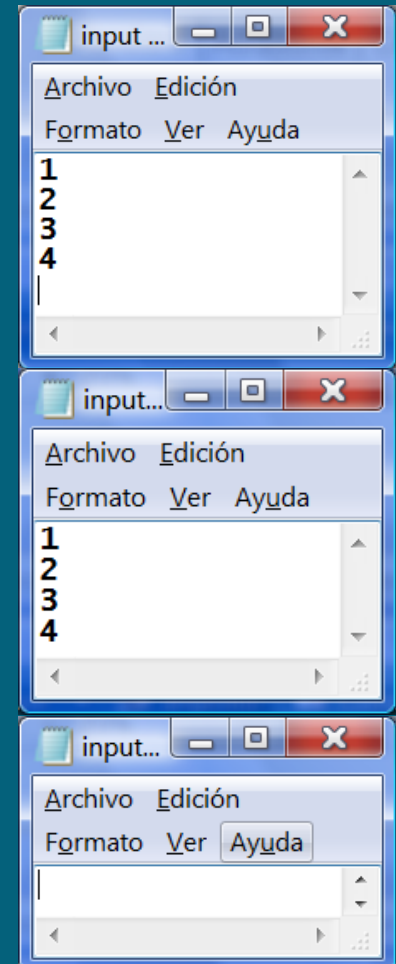
Mostrar la suma de los números reales que hay en el archivo *input.txt*.

```
double d, suma = 0;      ifstream archivo;

archivo.open("input.txt");
if (archivo.is_open()) {
    while (archivo >> d)
        suma = suma + d;
    cout << "Suma = " << suma << endl;
    archivo.close();
}
```

El operador >> notifica el fallo al bucle cuando no hay nada que leer y éste se detiene.

Solución “universal” (válida tanto con archivos bien formateados como con aquellos que no lo están)



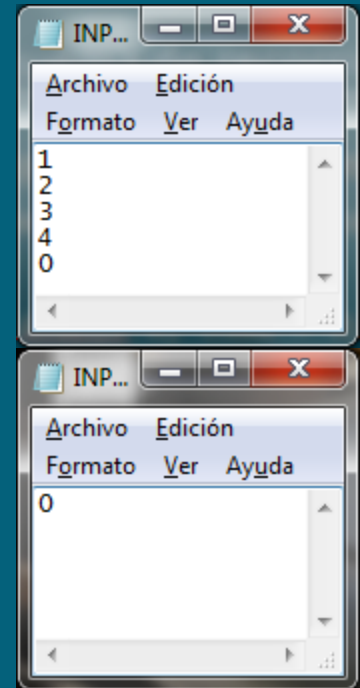
Esquema de recorrido

Ejemplo 4.- Secuencia explícita leída de archivo

Mostrar la suma de los números reales que hay en el archivo *input.txt* que finaliza con valor **centinela** 0.

```
double d, suma = 0;    ifstream archivo;

archivo.open("input.txt");
if (archivo.is_open()) {
    archivo >> d;
    while (d != 0) {
        suma = suma + d;
        archivo >> d;
    }
    cout << "Suma = " << suma << endl;
    archivo.close();
}
```



Esquema de recorrido

Ejemplo 4bis.- Secuencia explícita leída de archivo

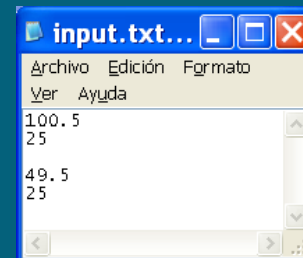
Suma y media de los números reales que hay en un archivo.

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

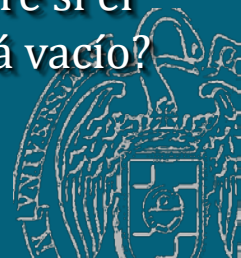
int main()
{
    double d, suma = 0;    int cont = 0;    ifstream archivo;

    archivo.open("input.txt"); // Apertura
    if (!archivo.is_open()) cout << "ERROR DE APERTURA DE ARCHIVO" << endl;
    else {
        archivo >> d;
        while (!archivo.eof()) {
            suma = suma + d;    cont = cont + 1;
            archivo >> d;
        }
        cout << "Suma = " << suma << endl;
        cout << "Media = " << suma / cont << endl;
        archivo.close(); // Cierre del archivo
    }
    return 0;
}
```

sumamedia1.cpp



¿Qué ocurre si el fichero está vacío?



Esquema de recorrido

Ejemplo 4bis.- Secuencia explícita leída de archivo

Suma y media de los números reales que hay en un archivo.

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

```
int main()
{
```

```
    double d, suma = 0; int cont = 0; ifstream archivo;
```

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

```
    if (!archivo.is_open()) cout << "ERROR DE APERTURA DE ARCHIVO" << endl;
```

```
    else {
```

```
        while (archivo >> d) {
```

```
            suma = suma + d;
```

```
            cont = cont + 1;
```

```
        }
```

```
        cout << "Suma = " << suma << endl;
```

```
        cout << "Media = " << suma / cont << endl;
```

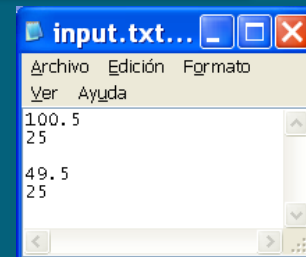
```
        archivo.close(); // Cierre del archivo
```

```
    }
```

```
    return 0;
```

```
}
```

sumamedia2.cpp



¿Qué ocurre si el
fichero está vacío?

Esquema de recorrido

Secuencias calculadas (recurrentes) e_1, e_2, e_3, \dots

$$e_1 = v_1$$
$$e_i = f(i, e_{i-1}) \quad i \geq 2$$

Ejemplo:

$$e_1 = 1$$
$$e_{i+1} = e_i + 1 \quad (i \geq 2)$$

1 2 3 4 5 6 7 8 ...

$$e_1 = v_1$$

$$e_2 = v_2$$

$$e_i = f(i, e_{i-1}, e_{i-2}) \quad i \geq 3$$

Ejemplo: Los números de Fibonacci

$$F_1 = 0$$

$$F_2 = 1$$

$$F_i = F_{i-1} + F_{i-2} \quad (i \geq 3)$$

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

Calcular el n-ésimo término, suma o multiplicación de x términos, ...



Esquema de recorrido

sumatorio.cpp

Ejemplo 5.- Secuencias calculadas

Calcular la suma de los N primeros números naturales (1 + 2 + 3 + ...) (N dado por el usuario)

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

```
int N;
cout << "Introduzca el numero natural tope N = ";
cin >> N;
if (N <= 0) cout << "El numero debe ser positivo";
else {
    int i = 1;
    int sumatorio = 0;
    while (i <= N) {
        sumatorio = sumatorio + i;
        i++;
    }
    cout << "Sumatorio de 1 a " << N << " = " << sumatorio;
}
```



Esquema de recorrido

Ejemplo 6.- Secuencias calculadas

fibonacci.cpp

Mostrar los números de Fibonacci menores o iguales que un número N dado por el usuario

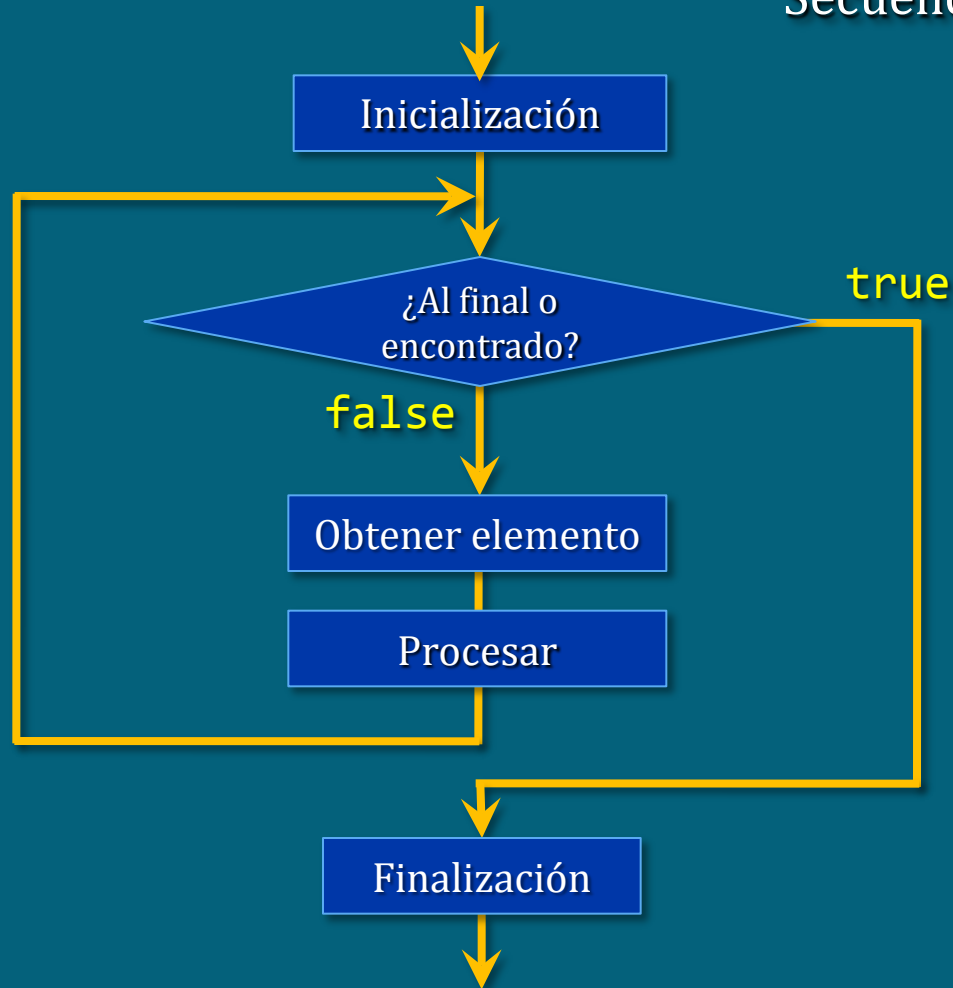
Si N es 50, la secuencia será: 0 1 1 2 3 5 8 13 21 34

```
int num, fib, fibMenos2 = 0, fibMenos1 = 1;
cout << "Hasta: "; cin >> num;
if (num >= 1) {
    cout << "0 1 "; // Los dos primeros seguro que son <= num
    fib = fibMenos2 + fibMenos1;
    while (fib <= num) {
        cout << fib << " ";
        fibMenos2 = fibMenos1;
        fibMenos1 = fib;
        fib = fibMenos2 + fibMenos1;
    }
}
```



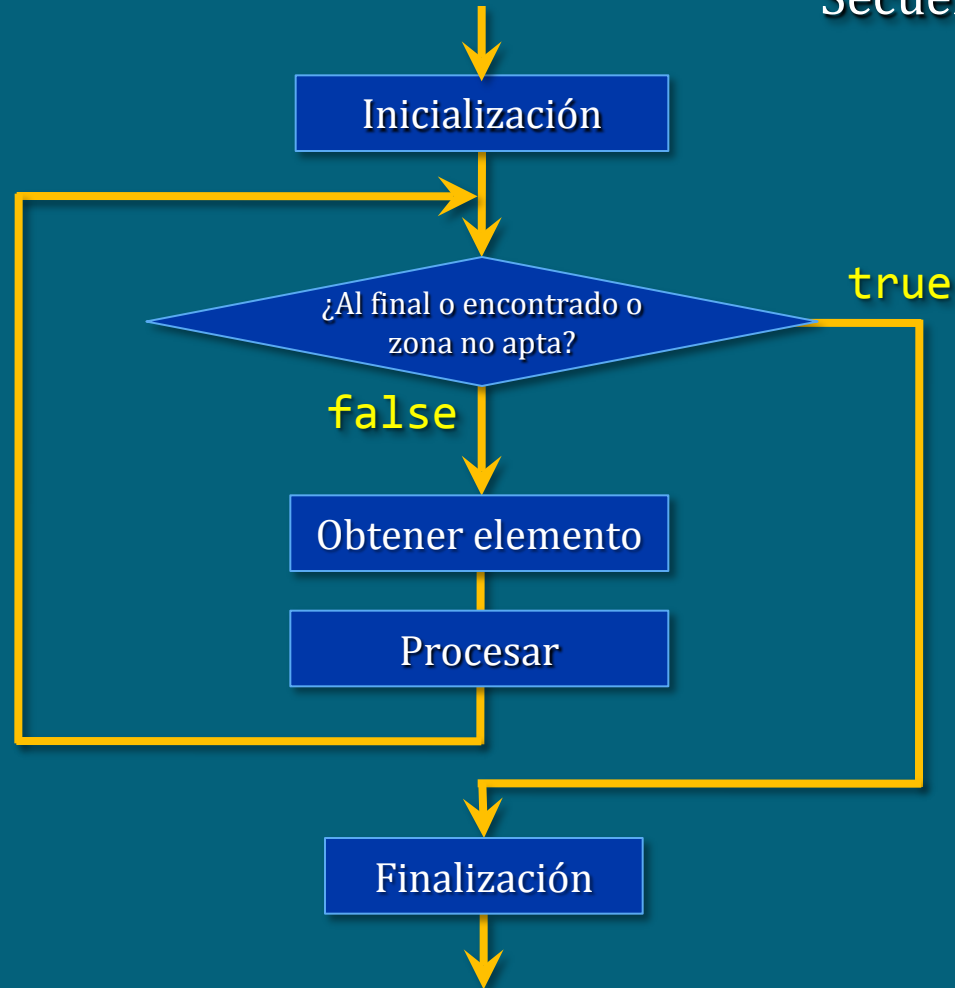
Esquema de búsqueda

Secuencia no ordenada



Esquema de búsqueda

Secuencia ordenada



Esquema de búsqueda

Ejemplo 6.- Secuencia explícita leída del teclado (con centinela)

Número de caracteres anteriores a la aparición de * en una cadena de caracteres acabada en .

```
const char final = '.';   const char buscado = '*';
char ch;      int cont = 0;
cout << "Introduce una cadena acabada en punto:" << endl;
cin.get(ch);
while ((ch != final) && (ch != buscado)) {
    cont++;
    cin.get(ch);
}
if (ch == buscado) {
    cout << buscado << " ha sido encontrado" << endl;
    cout << "Habia " << cont << " caracteres antes";
} else
    cout << buscado << " no ha sido encontrado" << endl;
```



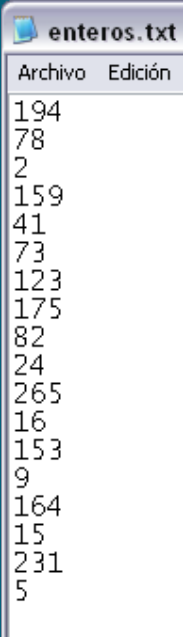
Esquema de búsqueda

Ejemplo 7.- Secuencia explícita leída de archivo

Localizar elemento en un archivo (no ordenado).

busca_elemento_fich.cpp

```
int i, buscado;
int cont = 0;
bool encontrado = false;
ifstream archivo;
archivo.open("enteros.txt");
cout << "Valor a localizar: "; cin >> buscado;
archivo >> i;
while (!archivo.eof() && !encontrado) {
    cont++;
    encontrado = i == buscado;
    archivo >> i;
}
if (encontrado)
    cout << "Encontrado (pos.: " << cont << ")";
else cout << "No encontrado";
archivo.close();
```



Archivo	Edición
194	
78	
2	
159	
41	
73	
123	
175	
82	
24	
265	
16	
153	
9	
164	
15	
231	
5	



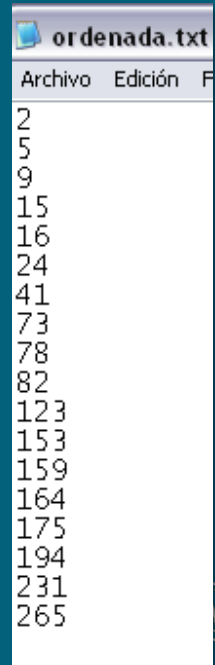
Esquema de búsqueda

Ejemplo 8.- Secuencia explícita leída de archivo

Localizar elemento en un archivo ordenado.

```
int i, buscado;
int cont = 0;
bool encontrado = false, encontradoMayor = false;
ifstream archivo;

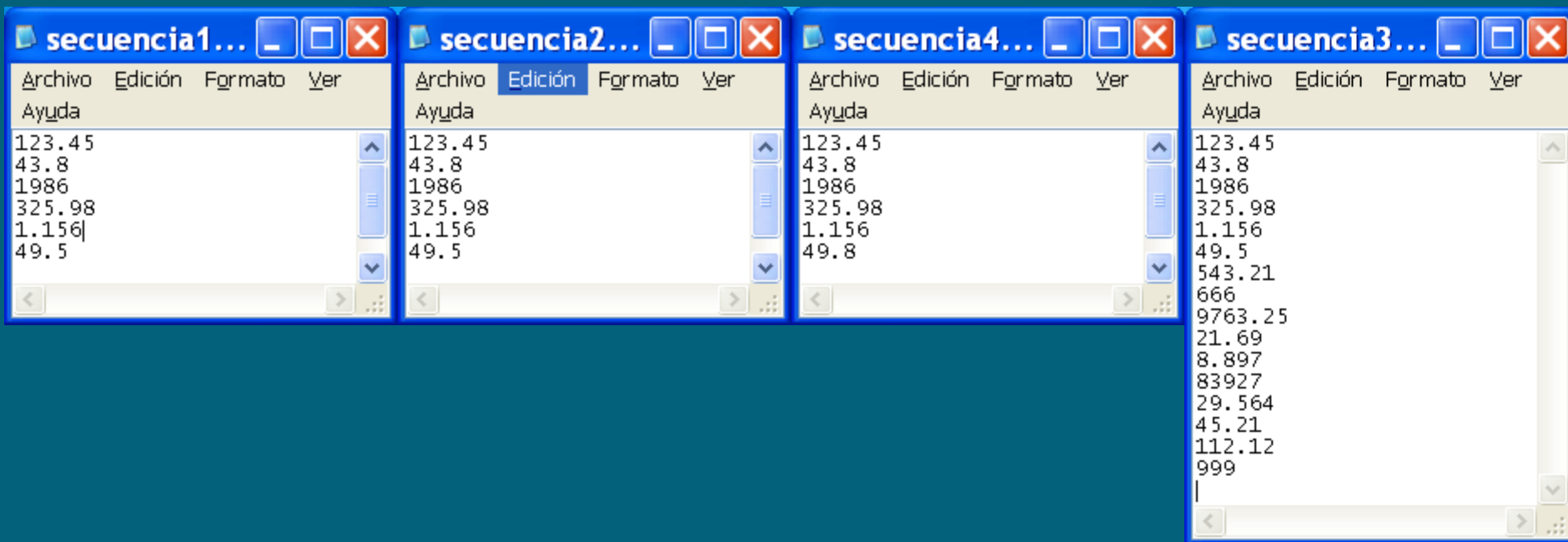
archivo.open("ordenada.txt");
cout << "Valor a localizar: "; cin >> buscado;
archivo >> i;
while (!archivo.eof() && !encontrado && !encontradoMayor) {
    cont++;
    encontrado = i == buscado;
    encontradoMayor = i > buscado;
    archivo >> i;
}
if (encontrado)
    cout << "Encontrado (pos.: " << cont << ")";
else {
    cout << "No encontrado" << endl;
    if (!encontradoMayor) cout << "Todos eran menores";
    else cout << "Y los hay mayores";
}
archivo.close();
```



Esquema de búsqueda

Ejemplo 9.- Secuencia explícita leída de archivo

¿Son iguales dos archivos?



The image displays four Notepad windows, each containing a list of numbers. The windows are titled 'secuencia1...', 'secuencia2...', 'secuencia4...', and 'secuencia3...'. The first three windows show identical sequences of numbers: 123.45, 43.8, 1986, 325.98, 1.156, and 49.5. The fourth window, 'secuencia3...', shows a longer sequence: 123.45, 43.8, 1986, 325.98, 1.156, 49.5, 543.21, 666, 9763.25, 21.69, 8.897, 83927, 29.564, 45.21, 112.12, and 999.

Window Title	Content
secuencia1...	123.45 43.8 1986 325.98 1.156 49.5
secuencia2...	123.45 43.8 1986 325.98 1.156 49.5
secuencia4...	123.45 43.8 1986 325.98 1.156 49.8
secuencia3...	123.45 43.8 1986 325.98 1.156 49.5 543.21 666 9763.25 21.69 8.897 83927 29.564 45.21 112.12 999



Esquema de búsqueda

Ejemplo 9.- Secuencia explícita leída de archivo

¿Son iguales dos archivos?

```
double d1, d2;  
ifstream sec1, sec2;  
bool iguales = true; // 2 secuencias vacías son iguales  
sec1.open("secuencia1.txt");  
sec2.open("secuencia2.txt");
```



Completa el programa

```
sec1.close();  
sec2.close();
```



Esquema de búsqueda

Ejemplo 10.- Secuencia explícita leída del teclado (con centinela)

¿Paréntesis bien emparejados?

ab(c(de) fgh)#

ab(c(de)) fgh#

ab(c(de) fgh((i(jk))lmn)op)#



ab(c(de) fgh#

ab(c(de))) fg(h#



Esquemas mixtos

Ejemplo 11.- Secuencia explícita leída de archivo

Contabilizar el número de elementos iguales a uno dado en un archivo ordenado (crecientemente)

¿Algoritmo?



¿Programa?



Esquemas mixtos

Ejemplo 12.- Secuencia explícita leída de teclado (con centinela)

Mostrar por pantalla una cadena de caracteres introducida por teclado eliminando los espacios en blanco que dicha cadena pueda contener al principio. La cadena acaba en fin de línea.

¿Algoritmo?



¿Programa?



Esquemas mixtos

Ejemplo 12.- Secuencia explícita leída de teclado (con centinela)

Mostrar por pantalla una cadena de caracteres introducida por teclado eliminando los espacios en blanco que dicha cadena pueda contener al principio. La cadena acaba en fin de línea.

```
char c;
cout << "Introduzca una línea de texto acabada en Intro..." << endl;
cin.get(c);

//Descartar los blancos iniciales
while (c != '\n' && c == ' ')
    cin.get(c);
//Recorrer la "cola" de la cadena escribiéndola tal cual es
while (c != '\n') {
    cout << c;
    cin.get(c);
}
// Escribir el salto de línea
cout << endl;
```



Esquemas mixtos

Ejemplo 13.- Secuencias explícitas leídas de archivo

Generar un fichero de texto a partir de otro conteniendo el fichero destino el mismo número de líneas que el fichero origen.

El contenido de cada línea del fichero destino será igual al de la correspondiente línea del fichero origen salvo que se habrán eliminado los espacios en blanco que pudiese haber al principio.

¿Algoritmo?

¿Programa?



Tipos definidos por el programador: el caso de los enumerados

Clasificación de los tipos de datos

✓ Simples

- ❖ Estándar: **int**, **float**, **double**, **char**, **bool**
El conjunto de valores está predeterminado.
- ❖ Definidos por el usuario: *enumerados*
El conjunto de valores lo define el programador.

✓ Estructurados

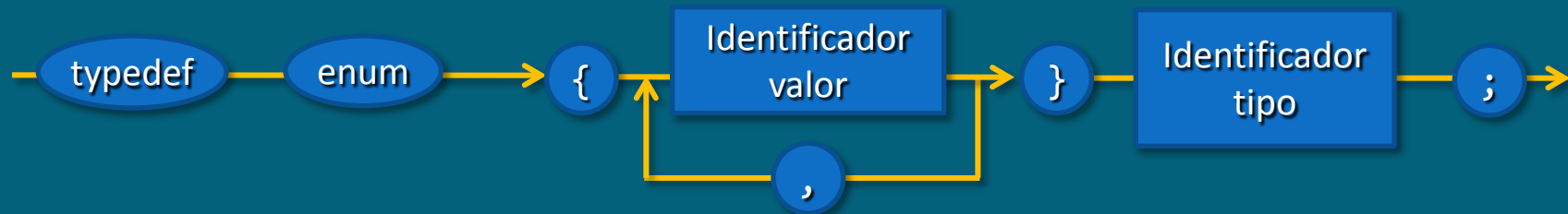
- ❖ Colecciones homogéneas: *arrays* y *archivos de texto*
Todos los elementos de la colección son del mismo tipo.
- ❖ Colecciones heterogéneas: *estructuras*
Los elementos de la colección pueden ser de tipos distintos.



Enumerados

Definición del tipo

Cada valor se identifica con un *símbolo* (identificador).
Internamente los valores se almacenan como enteros.
Los tipos enumerados aumentan la legibilidad del código.



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



Enumerados

Declaración de variables

```
tMoneda moneda1, moneda2, moneda3;
```

```
moneda1 = cinco_centimos;
```

```
moneda2 = medio_euro;
```

```
moneda3 = moneda1;
```



Enumerados

E/S para variables de tipos enumerados

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

No se pueden leer o mostrar los valores usando los literales.

¿Qué ocurre con `cin >> mes;` ?

Si el usuario introduce por teclado **enero** o **junio** no se guardan los valores.
Se espera un valor entero.

¿Qué ocurre con `cout << mes;` ?

Se verá un número entero (el valor que se guarda internamente).

Necesitamos rutinas de E/S específicas.



Enumerados

Lectura del valor de una variable de tipo enumerado

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

```
int valorMes;      tMes mes;
```

```
cout << "1 - Enero" << endl;
cout << "2 - Febrero" << endl;
// resto de casos
cout << "12 - Diciembre" << endl;
do {
    cout << "Numero del mes...";      cin >> valorMes;
} while (valorMes < 1 || valorMes > 12);
```

```
switch (valorMes) {
case 1: mes = enero; break;
case 2: mes = febrero; break;
// resto de casos
case 12: mes = diciembre; break;
}
```



```
mes = tMes(valorMes - 1);
```



Enumerados

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

```
tMes leeMes();
```

```
int main() {  
    tMes mes;  
    ...  
    mes = leeMes();  
    ...  
    return 0;  
}
```

```
tMes leeMes() {  
    int valorMes;    tMes mesLeido;  
  
    cout << "1 - Enero" << endl;  
    cout << "2 - Febrero" << endl;  
    // resto de casos  
    cout << "12 - Diciembre" << endl;  
    do {  
        cout << "Numero del mes...";    cin >> valorMes;  
    } while (valorMes < 1 || valorMes > 12);  
    mesLeido = tMes(valorMes - 1);  
    return mesLeido;  
}
```



Si los tipos se usan en varias funciones, los declaramos en la sección de declaraciones globales antes de los prototipos



Enumerados

Escritura del valor de una variable de tipo enumerado

```
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 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  
}
```



Implementa una función que devuelva una cadena equivalente al

mes que recibe

Página 77



Enumerados

Ordenación de los valores del tipo

Los enumerados son tipos ordenados.

Los valores quedan ordenados de acuerdo con su posición al declarar el tipo.

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

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

```
tDia dia;
```

```
bool laborable;
```

```
if (dia == domingo)    cout << "Hoy es festivo" << endl;
```

```
laborable = (dia >= lunes) && (dia <= viernes);
```



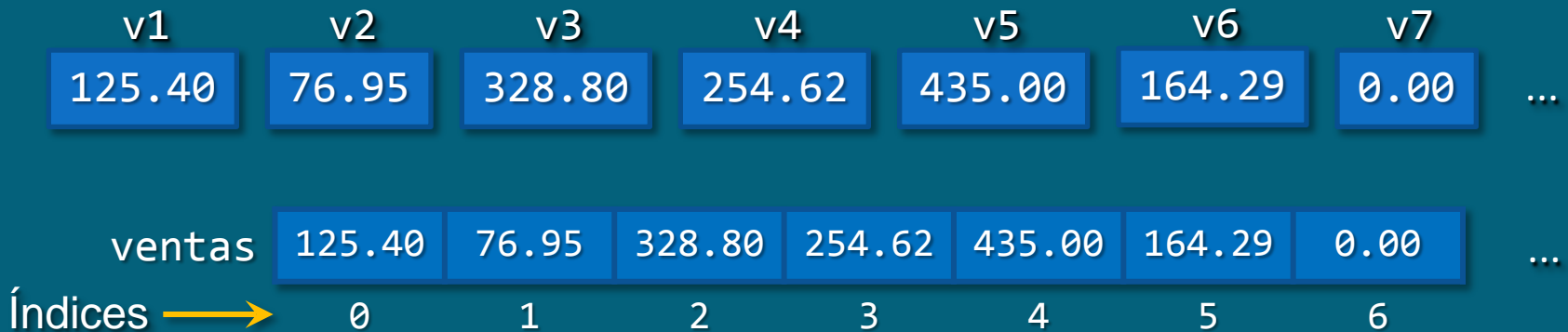
Tipos definidos por el programador: el caso de los arrays



Arrays

Definición del tipo

Colección homogénea de elementos *versus* variables “sueltas”



El tipo de los elementos (el tipo base) puede ser cualquiera estándar o uno definido por el programador previamente

```
typedef tipo_base nombre_tipo_array[tamaño];
```

```
typedef double tVentas[31];
```

```
typedef double tTemp[7];
```

```
typedef tMoneda tCalderilla[15]; // Enumerado tMoneda
```



Arrays

Declaración de variables

nombre_tipo_array nombre_var;

```
tTemp tempMax;
```

tempMax

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



NO se inicializan los elementos automáticamente



Arrays

Acceso a los elementos de un array

nombre_var [indice]

```
typedef double tVentas[31];  
tVentas ventas;
```

```
ventas[0] = 125.40;  
cout << ventas[30];  
if (ventas[0] == 1000) ...
```

```
ventas[-1]  
ventas[31]  
ventas[40]
```



No hay comprobación automática de la validez del índice

¡Es responsabilidad del programador!



Arrays

Ejemplo.- Cálculo de la media de un array de temperaturas de la semana

```
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;
```



Define los tamaños de los arrays con constantes



Arrays

Ejemplo.- Cálculo de la media de un array de temperaturas de la semana (versión usando función)

```
#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



Arrays

Ejemplo.- Cálculo de la media de un array de temperaturas de la semana (versión usando función) (cont.)

```
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, cuando son datos de entrada, se pasan como parámetros constantes

Las funciones no pueden devolver arrays



Arrays

Ejemplo.- Localización del primer día del año en el que las ventas superan 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)
        if (ventas[ind] > 1000)
            encontrado = true;
        else
            ind = ind + 1;
    if (!encontrado)
        ind = -1;

    return ind;
}
```



Arrays

Ejemplo.- Copia de un array en otro

```
const int N = 100;  
typedef int TArray[N];  
TArray array1, array2;
```

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];
```



Contenidos extra del tema

Conversión de tipos

Moldes (casts)

Para forzar 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;
```

```
a / b // evalúa a 1 (división entera)
```

```
double(a) / b // evalúa a 1.5 (división real)
```

Operadores (prioridad)	Asociatividad
++ -- (postfijos) Llamadas a funciones Moldes	Izda. a dcha.
++ -- (prefijos) ! - (cambio de signo)	Dcha. a izda.
* / %	Izda. a dcha.

...



Conversión de tipos

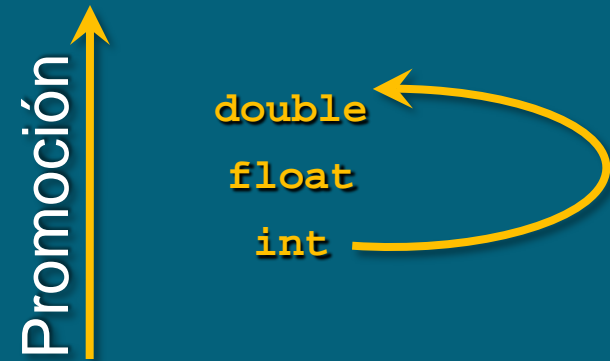
Conversiones automáticas de tipos

Promoción de tipos:

- Si los dos operandos numéricos de una operación binaria son de tipos distintos, el valor del tipo *menor* se promociona al tipo *mayor* antes de operar.

Se convierten los valores, **NO** las variables.

```
int j = 2, i = 3;  
double a = 1.5;  
a + i * j;  
a + 3 * 2;  
1.5 + 6;
```

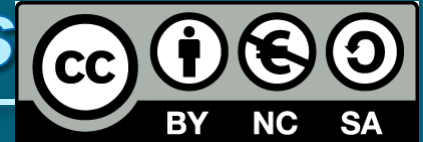


Se convierte el valor **6 int** (4 bytes) en **double** (8 bytes)

¡Las variables no pueden cambiar de tipo!






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.

En <http://es.creativecommons.org/> y <http://creativecommons.org/> puedes saber más de Creative Commons.

