

3

Programación Modular

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

Luis Hernández Yáñez
Facultad de Informática
Universidad Complutense



Índice

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



Programas multiarchivo y compilación separada



Programación modular

Programas multiarchivo

Código fuente repartido entre varios archivos (*módulos*)

Cada módulo con sus declaraciones y sus subprogramas

→ Módulo: Unidad funcional (estructura de datos, utilidades, ...)

Lista

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elem;
    int cont;
} tArray;

void init(tArray &lista);

void insert(tArray &lista,
            double elem, bool &ok);

void remove(tArray &lista,
            int pos, bool &ok);
...
```

Principal

```
int main() {
    tArray lista;
    bool ok;
    init(lista);
    cargar(lista, "bd.txt");
    sort(lista);
    double dato;
    cout << "Dato: ";
    cin >> dato;
    insert(lista, dato, ok);
    cout << min(lista) << endl;
    cout << max(lista) << endl;
    cout << sum(lista) << endl;
    guardar(lista, "bd.txt");

    return 0;
}
```

Cálculos

```
double mean(tArray lista);
double min(tArray lista);
double max(tArray lista);
double dev(tArray lista);
int minIndex(tArray lista);
int maxIndex(tArray lista);
double sum(tArray lista);
```

Archivos

```
bool cargar(tArray &lista,
            string nombre);

bool guardar(tArray lista,
             string nombre);

bool mezclar(string arch1,
              string arch2);

int size(string nombre);

bool exportar(string nombre);
```

Ejecutable



Programación modular

Compilación separada

Cada módulo se compila a código objeto de forma independiente

Lista

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elem;
    int cont;
} tArray;

void init(tArray &lista);

void insert(tArray &lista,
            double elem, bool &ok);

void remove(tArray &lista,
            int pos, bool &ok);
...
```

lista.obj

```
00101110101011001010010010101
0010101001010101111010101000
101001010101010101010101010101
011001010101010101010101010101
01001010101010000101010101101
010010101010101000010101010111
0100101010101010100001010101011
110010101010101110011001010101
01101010101010010010010101010111
0010101010100101010101010101010
101001010101010101010101011100
10010101011001010101010101010100
1010101010101010101010101010101
01000010101011100101010101010100
0110101010111010011010101010101
010111111010101010010101010111
00001001010100101010101010101010
```

Cálculos

```
double mean(tArray lista);
double min(tArray lists);
double max(tArray lista);
double desv(tArray lista);
int minIndex(tArray lista);
int maxIndex(tArray lista);
double sum(tArray lista);
```

calculos.obj

```
0101100101001010101001010100
1010101111010101000101010101
1010100010101010101010101010
1010101010101010101010101010
101000001010101010101010101
0101010000101010111001010101
0101011110011001010101010101
01010010010101010111001010101
0100101010101010101010101010
10100101000010011110100101010
110010101010101010010101010101
010100101010101010101010000101
010111001010101000111010101010
1110100110101010101011100001010
101010101010101010101010101010
```

Archivos

```
bool cargar(tArray &lista,
            string nombre);
bool guardar(tArray lista,
             string nombre);
bool mezclar(string arch1,
              string arch2);
int size(string nombre);
bool exportar(string nombre);
```

archivos.obj

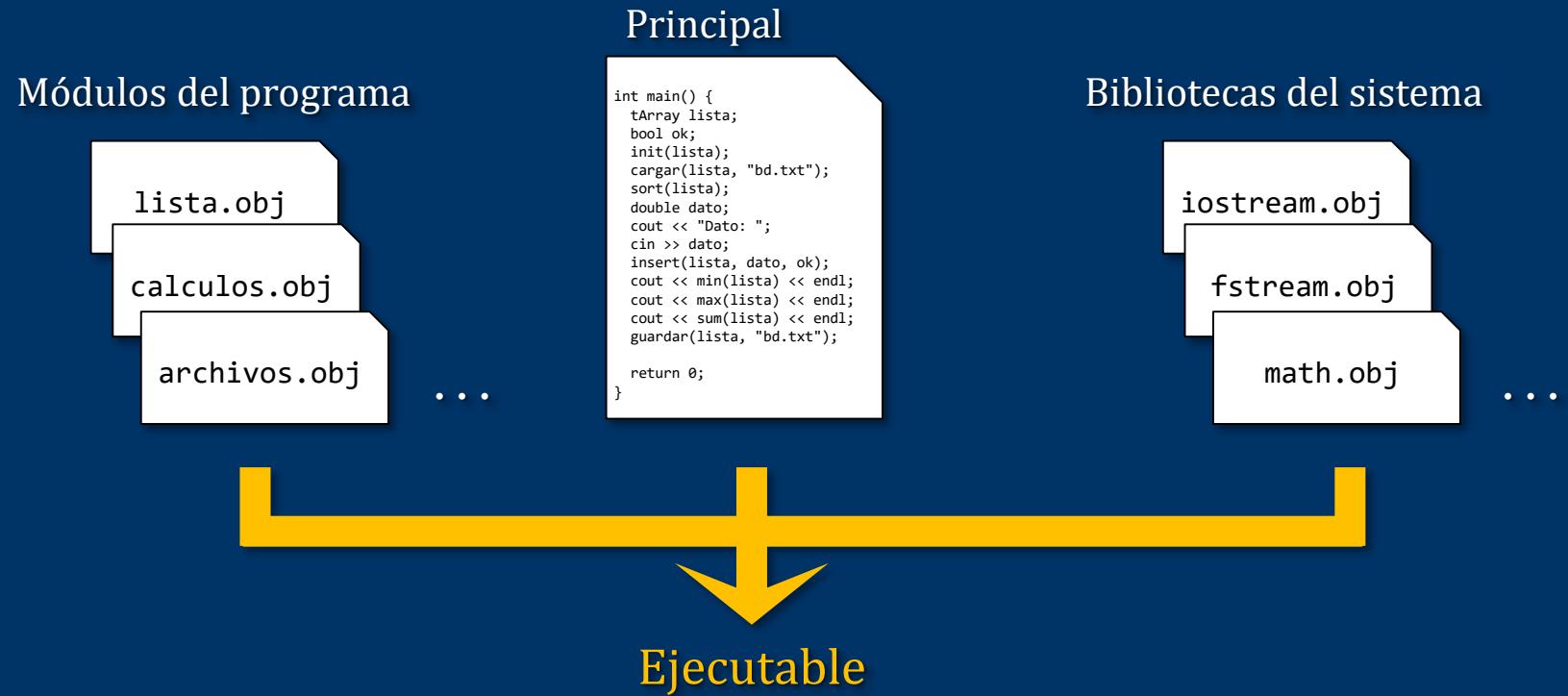
```
111010101100101001001010101010
1010010101011110101010001010
01010101010101010101010101010
01010101010101010101010101010
01010101000001010101011010100
101010101010101010101010101010
101010101010101010101010101010
101010101010101010101010101010
101010101010101010101010101010
01010101000100111101001
01010110010101010101010101010
101010101010101010101010101010
00101010111001010101010101010
01010111010101010101010101010
01010111010101010101010101010
111010101010101010101010101010
1001010101010101010101010101010
```



Programación modular

Compilación separada

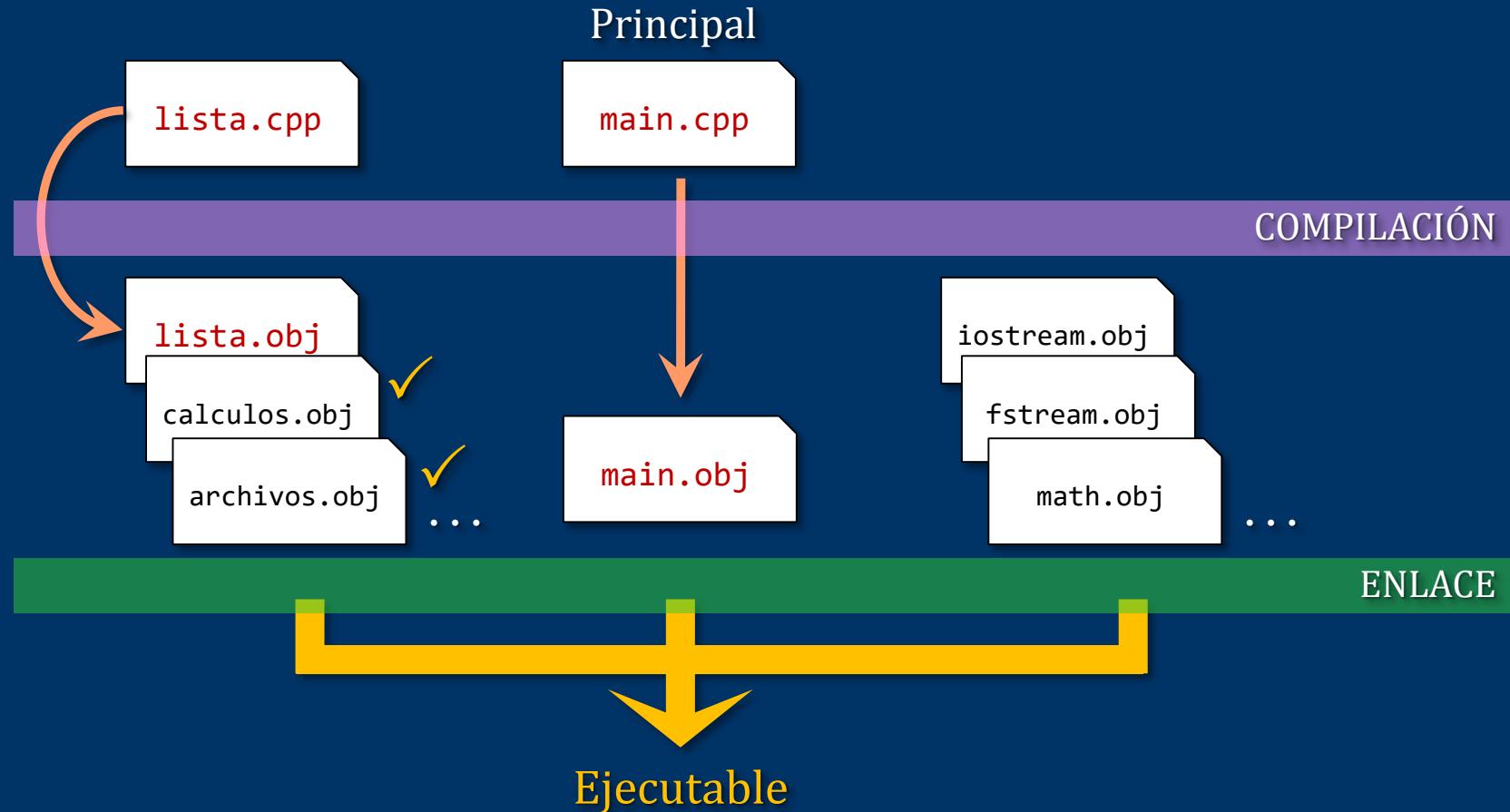
Al compilar el programa principal, se adjuntan los módulos compilados



Programación modular

Compilación separada

¡Sólo los archivos fuente modificados necesitan ser recompilados!



Interfaz frente a implementación



Interfaz frente a implementación

Creación de módulos de biblioteca

Código de un programa de un único archivo:

- ✓ Definiciones de constantes
- ✓ Declaraciones de tipos de datos
- ✓ Prototipos de los subprogramas
- ✓ Implementación de los subprogramas
- ✓ Implementación de la función `main()`

Constantes, tipos y prototipos indican *cómo se usa*: **Interfaz**

- ✓ Estructura de datos con los subprogramas que la gestionan
 - ✓ Conjunto de utilidades (subprogramas) de uso general
 - ✓ Etcétera
- + **Implementación** de los subprogramas (*cómo se hace*)



Interfaz frente a implementación

Creación de módulos de biblioteca

Interfaz: Definiciones/declaraciones de datos y prototipos

¡Todo lo que el usuario de la unidad funcional necesita saber!

Implementación: Código de los subprogramas que hacen el trabajo

No hay que conocerlo para usarlo: ¡Seguro que es correcto!

Interfaz e implementación en dos archivos separados:

- ✓ Cabecera: Definiciones/declaraciones de datos y prototipos
- ✓ Implementación: Implementación de los subprogramas.

Archivo de cabecera: extensión .h

Archivo de implementación: extensión .cpp } Mismo nombre

Repartimos el código entre ambos archivos (`lista.h/lista.cpp`)



Interfaz frente a implementación

Creación de módulos de biblioteca

Interfaz frente a implementación

lista.h

```
const int N = 10;
typedef double tArray[N];
typedef struct {
    tArray elem;
    int cont;
} tArray;

void init(tArray &lista);
void insert(tArray &lista,
            double elem, bool &ok);
void remove(tArray &lista,
            int pos, bool &ok);
...
```

lista.cpp

```
#include "lista.h"

void init(tArray &lista) {
    lista.cont = 0;
}

void insert(tArray &lista,
            double elem, bool &ok) {
    if (lista.cont == N) {
        ok = false;
    }
    else {
        ...
    }
}
```

Módulo
Unidad
Biblioteca

Si otro módulo quiere usar algo de esa biblioteca:
Debe incluir el archivo de cabecera

main.cpp

```
#include "lista.h"
...
```

Los nombres de archivos de cabecera propios (no del sistema) se encierran entre dobles comillas, no entre ángulos



Interfaz frente a implementación

Creación de módulos de biblioteca

Interfaz

Archivo de cabecera (.h): todo lo que necesita conocer otro módulo (o programa principal) que quiera utilizar sus servicios (subprogramas)

La directiva **#include** añade las declaraciones del archivo de cabecera en el código del módulo (*preprocesamiento*):

main.cpp

```
#include "lista.h"  
..."
```

Preprocesador



Todo lo que se necesita saber para comprobar si el código de main.cpp hace un uso correcto de la lista (declaraciones y llamadas)

lista.h

```
const int N = 10;  
typedef double tArray[N];  
typedef struct {  
    tArray elem;  
    int cont;  
} tArray;  
  
void init(tArray &lista);  
  
void insert(tArray &lista,  
           double elem, bool &ok);  
  
void remove(tArray &lista,  
            int pos, bool &ok);  
..."
```

main.cpp

```
const int N = 10;  
typedef double tArray[N];  
typedef struct {  
    tArray elem;  
    int cont;  
} tArray;  
  
void init(tArray &lista);  
  
void insert(tArray &lista, double elem,  
           bool &ok);  
  
void remove(tArray &lista, int pos,  
            bool &ok);  
..."
```



Interfaz frente a implementación

Creación de módulos de biblioteca

Implementación

Compilar el módulo significa compilar su archivo de implementación (.cpp)

También necesita conocer sus propias declaraciones:

lista.cpp

```
#include "lista.h"  
...
```

lista.cpp

```
#include "lista.h"  
  
void init(tArray &lista) {  
    lista.cont = 0;  
}  
  
void insert(tArray &lista,  
double elem, bool &ok) {  
    if (lista.cont == N) {  
        ok = false;  
    }  
    else {  
        ...  
    }  
}
```

lista.obj

```
00101110101011001010010010101  
0010101001010101111010101000  
10100101010101010100101010101  
01100101010101010101010101001  
011010101010100000101010101101  
01001010101010101000101010111  
11001010101010111000110010101  
01101010101010100101010101111  
001010101010101010101010101010  
101001010101010000101011110  
100101010101100101010101010100  
10101010101010101010101010101  
01000010101011100010101010100  
0111010101110100110101010101  
01011111110101011001101010111  
00001001010101001010101010110
```

Al compilar el módulo se genera el código objeto

Si no se modifica no hay necesidad de recompilar

Código que usa el módulo:

- ✓ Necesita sólo el archivo de cabecera para compilar
- ✓ Se adjunta el código objeto del módulo durante el enlace



Uso de módulos de biblioteca



Uso de módulos de biblioteca

Ejemplo: Gestión de una lista ordenada (Tema 7)

Todo lo que tenga que ver con la lista estará en su propio módulo

Ahora el código estará repartido en tres archivos:

- ✓ `lista.h`: archivo de cabecera del módulo de lista
- ✓ `lista.cpp`: implementación del módulo de lista
- ✓ `bd.cpp`: programa principal que usa la lista

Tanto `lista.cpp` como `bd.cpp` deben incluir al principio `lista.h`

Módulo propio: dobles comillas en la directiva `#include`
`#include "lista.h"`

Archivos de cabecera de bibliotecas del sistema: entre ángulos

Y no tienen necesariamente que llevar extensión `.h`



Módulo: Gestión de una lista ordenada I

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

const int N = 100;
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
const string BD = "bd.txt";
...
```

```
1 #include <string>
2 using namespace std;
3
4 const int N = 100;
5
6 // Estructura para los datos individuales de la lista:
7 typedef struct {
8     int codigo;
9     string nombre;
10    double sueldo;
11 } tRegistro;
12
13 // Array de registros:
14 typedef tRegistro tArray[N];
15
16 // Lista: array y contador
17 typedef struct {
18     tArray registros;
19     int cont;
20 } tLista;
21
22 // Constante global con el nombre del archivo de base de datos:
23 const string BD = "bd.txt";
24
25 // Muestra en una línea la información del registro proporcionado
// precedida por su posición en la lista.
26 void mostrar(int pos, tRegistro registro);
27
28 // Muestra la lista completa.
29 void mostrar(const tLista &lista);
30
31 // Operador relacional para comparar registros.
32 // Basado en el campo nombre.
33 bool operator>(tRegistro opIzq, tRegistro opDer);
34
35 // Operador relacional para comparar registros.
36 // Basado en el campo nombre.
37 bool operator<(tRegistro opIzq, tRegistro opDer);
38
39 // Lectura de los datos de un nuevo registro.
40 tRegistro nuevo();
```

¡Documenta bien el código!



Programación modular

```
void mostrar(int pos, tRegistro registro);
void mostrar(const tLista &lista);
bool operator>(tRegistro opIzq, tRegistro opDer);
bool operator<(tRegistro opIzq, tRegistro opDer);
tRegistro nuevo();
void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N
int buscar(tLista lista, string nombre);
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
```

Cada prototipo, con un comentario que explique su utilidad/uso
(Aquí se omiten por cuestión de espacio)



Módulo: Gestión de una lista ordenada I

```
#include <iostream>
#include <string>
using namespace std;
#include <fstream>
#include <iomanip>
#include "lista.h"

tRegistro nuevo() {
    tRegistro registro;
    cout << "Introduce el código: ";
    cin >> registro.codigo;
    cout << "Introduce el nombre: ";
    cin >> registro.nombre;
    cout << "Introduce el sueldo: ";
    cin >> registro.sueldo;
    return registro;
} ...
```



Programación modular

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        }
        // Insertamos en la posición i
        for (int j = lista.cont; j > i; j--) {
            // Desplazamos a la derecha
            lista.registros[j] = lista.registros[j - 1];
        }
        lista.registros[i] = registro;
        lista.cont++;
    }
}
```

```
} ...
```



Programación modular

```
void eliminar(tLista &lista, int pos, bool &ok) { // pos = 1..  
    ok = true;  
    if ((pos < 1) || (pos > lista.cont)) {  
        ok = false; // Posición inexistente  
    }  
    else {  
        pos--; // Pasamos a índice del array  
        for (int i = pos + 1; i < lista.cont; i++) {  
            // Desplazamos a la izquierda  
            lista.registros[i - 1] = lista.registros[i];  
        }  
        lista.cont--;  
    }  
}  
  
...
```



Módulo: Gestión de una lista ordenada I

```
#include <iostream>
using namespace std;
#include "lista.h"

int menu();

int main() {
    tLista lista;
    bool ok;
    int op, pos;
    cargar(lista, ok);
    if (!ok) {
        cout << "No se ha podido abrir el archivo!" << endl;
    }
    else {
        do {
            mostrar(lista);
            op = menu(); ...
```



Programación modular

```
if (op == 1) {
    tRegistro registro = nuevo();
    insertar(lista, registro, ok);
    if (!ok) {
        cout << "Error: Lista llena!" << endl;
    }
}
else if (op == 2) {
    cout << "Posición: ";
    cin >> pos;
    eliminar(lista, pos, ok);
    if (!ok) {
        cout << "Error: Posicion inexistente!" << endl;
    }
}
else if (op == 3) {
    string nombre;
    cin.sync();
    cout << "Nombre: ";
    cin >> nombre;
    int pos = buscar(lista, nombre);
    ...
}
```



Programación modular

```
    if (pos == -1) {
        cout << "No se ha encontrado!" << endl;
    }
    else {
        cout << "Encontrado en la posición " << pos << endl;
    }
}
} while (op != 0);
guardar(lista);
}
return 0;
}

int menu() {
cout << endl;
cout << "1 - Insertar" << endl;
cout << "2 - Eliminar" << endl;
cout << "3 - Buscar" << endl;
cout << "0 - Salir" << endl;
int op;
do {
    ...
}
```



Compilación de programas multiarchivo



Compilación de programas multiarchivo

G++

Archivos de cabecera e implementación en la misma carpeta

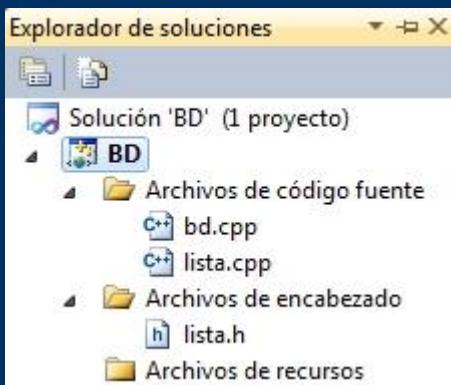
Listamos todos los .cpp en la orden g++:

```
D:\FP\Tema08>g++ -o bd.exe lista.cpp bd.cpp
```

Recuerda que sólo se compilan los .cpp

Visual C++/Studio

Archivos de cabecera e implementación en grupos distintos:



A los archivos de cabecera
los llama de encabezado

Con Depurar -> Generar solución
se compilan todos los .cpp



El preprocessador



El preprocesador

Directivas: # . . .

Antes de compilar se pone en marcha el *preprocesador*

Interpreta las directivas y genera un único archivo temporal con todo el código del módulo o programa

Como en la inclusión (directiva `#include`):

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

const int N = 100;

typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

typedef tRegistro
tArray[N];

typedef struct {
    tArray registros;
    int cont;
} tLista;
...
```

```
#include "lista.h"

int menu();
...
```

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

const int N = 100;

typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

typedef tRegistro
tArray[N];

typedef struct {
    tArray registros;
    int cont;
} tLista;
...

int menu();
...
```



Cada cosa en su módulo



Programación modular

Distribuir la funcionalidad del programa en módulos

Encapsulación de un conjunto de subprogramas relacionados:

- ✓ Por la estructura de datos sobre la que trabajan
- ✓ Subprogramas de utilidad

A menudo las estructuras de datos contienen otras estructuras:

```
const int N = 100;
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
```

Lista de registros:

- ✓ Estructura tRegistro
- ✓ Estructura tLista
(contiene tRegistro)

Cada estructura, en su módulo



Gestión de una lista ordenada II

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

typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

tRegistro nuevo();
bool operator>(tRegistro opIzq, tRegistro opDer);
bool operator<(tRegistro opIzq, tRegistro opDer);
void mostrar(int pos, tRegistro registro);
```



Gestión de una lista ordenada II

```
#include <iostream>
#include <string>
using namespace std;
#include <iomanip>
#include "registro.h"

tRegistro nuevo() {
    tRegistro registro;
    cout << "Introduce el código: ";
    cin >> registro.codigo;
    cout << "Introduce el nombre: ";
    cin >> registro.nombre;
    cout << "Introduce el sueldo: ";
    cin >> registro.sueldo;
    return registro;
}

bool operator>(tRegistro opIzq, tRegistro opDer) {
    return opIzq.nombre > opDer.nombre;
} ...
```



Gestión de una lista ordenada II

```
#include <string>
using namespace std;
#include "registro.h" ←

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
const string BD = "bd.txt";

void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N
int buscar(tLista lista, string nombre);
void mostrar(const tLista &lista);
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
```



Gestión de una lista ordenada II

```
#include <iostream>
using namespace std;
#include <fstream>
#include "lista2.h"
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        }
        // Insertamos en la posición i
        for (int j = lista.cont; j > i; j--) {// Desplazar a la derecha
            lista.registros[j] = lista.registros[j - 1];
        }
        ...
    }
}
```



Gestión de una lista ordenada II

```
#include <iostream>
using namespace std;
#include "registro.h" ←
#include "lista2.h" ←
int menu();

int main() {
    tLista lista;
    bool ok;
    int op, pos;

    cargar(lista, ok);
    if (!ok) {
        cout << "No se pudo abrir el archivo!" << endl;
    }
    else {
        do {
            mostrar(lista);
            op = menu();
            ...
        }
```



¡No intentes compilar este ejemplo!

Tiene errores



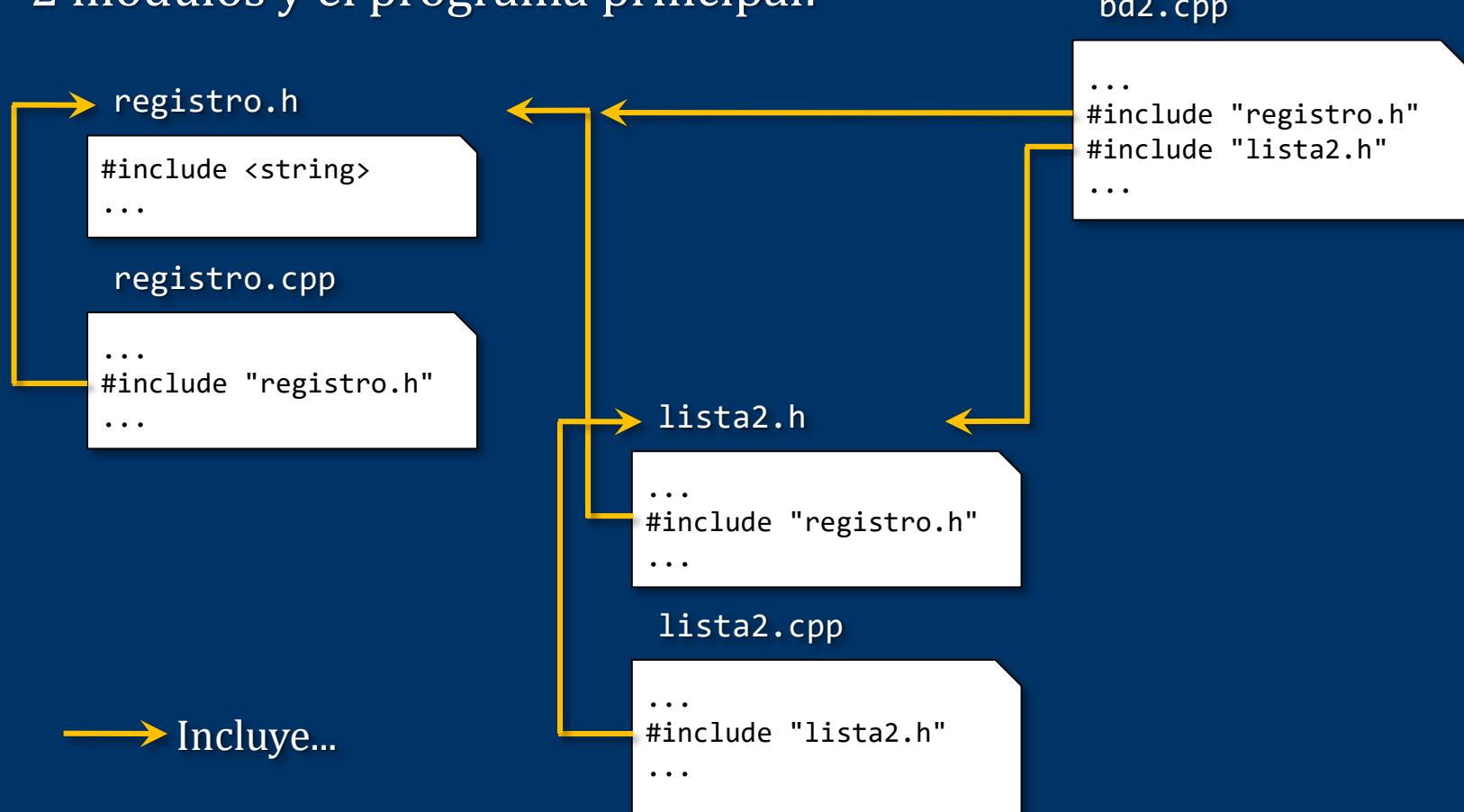
El problema de las inclusiones múltiples



Inclusiones múltiples

Gestión de una lista ordenada II

2 módulos y el programa principal:



Inclusiones múltiples

Gestión de una lista ordenada II

Preprocesamiento de #include:

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

```
#include "registro.h"
```

```
#include "lista2.h"
```

```
int menu();
```

```
...
```

```
#include <string>
using namespace std;
#include "registro.h"
```

```
const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
...
```

```
#include <string>
using namespace std;
typedef struct {
    ...
} tRegistro;
...
```

```
#include <string>
using namespace std;
typedef struct {
    ...
} tRegistro;
...
```



Inclusiones múltiples

Gestión de una lista ordenada II

Preprocesamiento de #include:

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

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

```
typedef struct {
    ...
} tRegistro;
...
```

```
#include "lista2.h"
```

```
int menu();
```

```
...
```



Sustituido

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

```
typedef struct {
    ...
} tRegistro;
...
```

```
const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
...
```



Inclusiones múltiples

Gestión de una lista ordenada II

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

#include <string>
using namespace std;

typedef struct {
    ...
} tRegistro; ←

...

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

typedef struct {
    ...
} tRegistro; ←

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
...

int menu();

...
```



Inclusiones múltiples

Compilación condicional

Directivas `#ifdef`, `#ifndef`, `#else` y `#endif`

Se usan en conjunción con la directiva `#define`

```
#define X  
#ifdef X  
... // Código if  
[#else  
... // Código else  
]  
#endif
```

```
#define X  
#ifndef X  
... // Código if  
[#else  
... // Código else  
]  
#endif
```

La directiva `#define` define un símbolo (identificador)

Izquierda: se compilará el “Código if” y no el “Código else”

Derecha: al revés, o nada si no hay else

Las cláusulas else son opcionales



Inclusiones múltiples

Protección frente a inclusiones múltiples

lista2.cpp y bd2.cpp incluyen registro.h

→ ¡Identificadores duplicados!

Cada módulo debe incluirse una y sólo una vez

Protección frente a inclusiones múltiples:

```
#ifndef X  
#define X  
... // Módulo  
#endif
```



El símbolo X debe ser único para cada módulo de la aplicación

La primera vez no está definido el símbolo X: se incluye y define

Las siguientes veces el símbolo X ya está definido: no se incluye

Símbolo X: nombre del archivo con _ en lugar de .

registro_h, lista2_h, ...



Gestión de una lista ordenada III

```
#ifndef registrofin_h
#define registrofin_h
#include <string>
using namespace std;

typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

tRegistro nuevo();
bool operator>(tRegistro opIzq, tRegistro opDer);
bool operator<(tRegistro opIzq, tRegistro opDer);
void mostrar(int pos, tRegistro registro);
#endif
```



Gestión de una lista ordenada III

```
#include <iostream>
#include <string>
using namespace std;
#include <iomanip>
#include "registrofin.h" ←————

tRegistro nuevo() {
    tRegistro registro;
    cout << "Introduce el código: ";
    cin >> registro.codigo;
    cout << "Introduce el nombre: ";
    cin >> registro.nombre;
    cout << "Introduce el sueldo: ";
    cin >> registro.sueldo;
    return registro;
}

bool operator>(tRegistro opIzq, tRegistro opDer) {
    return opIzq.nombre > opDer.nombre;
} ...
```



Gestión de una lista ordenada III

```
#ifndef listafin_h
#define listafin_h
#include <string>
using namespace std;
#include "registrofin.h" ←

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
const string BD = "bd.txt";
void mostrar(const tLista &lista);
void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N
int buscar(tLista lista, string nombre);
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
#endif
```



Gestión de una lista ordenada III

```
#include <iostream>
using namespace std;
#include <fstream>
#include "listafin.h" ←

void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        }
        // Insertamos en la posición i
        for (int j = lista.cont; j > i; j--) {
            // Desplazamos a la derecha
            lista.registros[j] = lista.registros[j - 1];
        }
        ...
    }
}
```



Gestión de una lista ordenada III

```
#include <iostream>
using namespace std;
#include "registrofin.h"    ←
#include "listafin.h"        ←

int menu();

int main() {
    tLista lista;
    bool ok;
    int op, pos;

    cargar(lista, ok);
    if (!ok) {
        cout << "No se pudo abrir el archivo!" << endl;
    }
    else {
        do {
            mostrar(lista);
            op = menu();
            ...
        }
    }
}
```



¡Ahora ya puedes compilarlo!



Inclusiones múltiples

Gestión de una lista ordenada III

Preprocesamiento de #include en bdfin.cpp:

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

#include "registrofin.h" ←
#include "listafin.h"

int menu();

...
```

```
#ifndef registrofin_h
#define registrofin_h
#include <string>
using namespace std;

typedef struct {
    ...
} tRegistro;
...
```



registrofin_h no se ha definido todavía



Inclusiones múltiples

Gestión de una lista ordenada III

Preprocesamiento de #include en bdfin.cpp:

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

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

```
typedef struct {
    ...
} tRegistro;
...
```

```
#include "listafin.h"
```

```
int menu();
```

```
...
```

```
#ifndef listafin_h
#define listafin_h
#include <string>
using namespace std;
#include "registrofin.h"
```

```
const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;
...
```



listafin_h no se ha definido todavía



Inclusiones múltiples

Gestión de una lista ordenada III

Preprocesamiento de #include en bdfin.cpp:

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

```
typedef struct {
    ...
} tRegistro;
...
```

```
#define listafin_h
#include <string>
using namespace std;
#include "registrofin.h"
...
int menu();
...
```

```
#ifndef registrofin_h
#define registrofin_h
#include <string>
using namespace std;

typedef struct {
    ...
} tRegistro;
...
```



¡registrofin_h ya está definido!



Implementaciones alternativas



Implementaciones alternativas

Misma interfaz, implementación alternativa

Lista
ordenada

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registro[i] <= registro))
            i++;
        // Insertamos en la posición i
        for (int j = lista.cont; j > i; j--) }
        // Desplazamos a la derecha
        lista.registros[j] = lista.registros[j - 1];
    ...
}
```

```
#include <string>
using namespace std;
#include "registrofin.h"

const int N = 100;
typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;

void insertar(tLista &lista, tRegistro registro, bool &ok);
```

lista.h

Lista
no ordenada

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        lista.registros[lista.cont] = registro;
        lista.cont++;
    }
}
```



Implementaciones alternativas

Misma interfaz, implementación alternativa

listaORD.cpp: Lista ordenada

```
...
#include "lista.h"

void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        }
        // Insertamos en la posición i
        for (int j = lista.cont; j > i; j--) {
            // Desplazamos a la derecha
            lista.registros[j] = lista.registros[j - 1];
        }
        lista.registros[i] = registro;
    ...
}
```

listaDES.cpp: Lista no ordenada

```
...
#include "lista.h"

void insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        lista.registros[lista.cont] = registro;
        lista.cont++;
    }
    ...
}
```



Implementaciones alternativas

Misma interfaz, implementación alternativa

Al compilar, incluimos un archivo de implementación u otro:

¿Programa con lista ordenada o con lista desordenada?

```
g++ -o programa.exe registrofin.cpp listaORD.cpp ...
```

Incluye la implementación de la lista con ordenación

```
g++ -o programa.exe registrofin.cpp listaDES.cpp ...
```

Incluye la implementación de la lista sin ordenación



Espacios de nombres



Espacios de nombres

Agrupaciones lógicas de declaraciones

Espacio de nombres: agrupación de declaraciones (tipos, datos, subprogramas) bajo un nombre distintivo

Forma de un espacio de nombres:

```
namespace nombre {  
    // Declaraciones  
}
```

Por ejemplo:

```
namespace miEspacio {  
    int i;  
    double d;  
}
```

Variables `i` y `d` declaradas en el espacio de nombres `miEspacio`



Espacios de nombres

Acceso a miembros de un espacio de nombres

Operador de resolución de ámbito (::)

Acceso a las variables del espacio de nombres `miEspacio`:

Nombre del espacio y operador de resolución de ámbito

`miEspacio::i`

`miEspacio::d`

Puede haber entidades con el mismo identificador en distintos módulos o en ámbitos distintos de un mismo módulo

Cada declaración en un espacio de nombres distinto:

```
namespace primero {           namespace segundo {  
    int x = 5;                 double x = 3.1416;  
}  
}
```

Ahora se distingue entre `primero::x` y `segundo::x`



Espacios de nombres

using

Introduce un nombre de un espacio de nombres en el ámbito actual:

```
#include <iostream>
using namespace std;
namespace primero {
    int x = 5;
    int y = 10;
}
namespace segundo {
    double x = 3.1416;
    double y = 2.7183;
}
int main() {
    using primero::x;
    using segundo::y;
    cout << x << endl; // x es primero::x
    cout << y << endl; // y es segundo::y
    cout << primero::y << endl; // espacio explícito
    cout << segundo::x << endl; // espacio explícito
    return 0;
}
```

5
2.7183
10
3.1416



Espacios de nombres

using namespace

Introduce todos los nombres de un espacio en el ámbito actual:

```
#include <iostream>
using namespace std;
namespace primero {
    int x = 5;
    int y = 10;
}
namespace segundo {
    double x = 3.1416;
    double y = 2.7183;
}
int main() {
    using namespace primero;
    cout << x << endl; // x es primero::x
    cout << y << endl; // y es primero::y
    cout << segundo::x << endl; // espacio explícito
    cout << segundo::y << endl; // espacio explícito
    return 0;
}
```

using [namespace]
sólo tiene efecto
en el bloque
en que se encuentra

5
10
3.1416
2.7183



Ejemplo de espacio de nombres

```
#ifndef listaEN_h
#define listaEN_h
#include "registrofin.h"

namespace ord { // Lista ordenada
    const int N = 100;
    typedef tRegistro tArray[N];
    typedef struct {
        tArray registros;
        int cont;
    } tLista;
    const string BD = "bd.txt";
    void mostrar(const tLista &lista);
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    void eliminar(tLista &lista, int pos, bool &ok); // 1..N
    int buscar(tLista lista, string nombre);
    void cargar(tLista &lista, bool &ok);
    void guardar(tLista lista);
} // namespace

#endif
```



Ejemplo de espacio de nombres

Implementación

```
#include <iostream>
#include <fstream>
using namespace std;
#include "listaEN.h"

void ord::insertar(tLista &lista, tRegistro registro, bool &ok) {
    // ...
}

void ord::eliminar(tLista &lista, int pos, bool &ok) {
    // ...
}

int ord::buscar(tLista lista, string nombre) {
    // ...
}

...
```



Ejemplo de espacio de nombres

Uso del espacio de nombres

Quien utilice listaEN.h debe poner el nombre del espacio:

```
#include <iostream>
using namespace std;
#include "registrofin.h"
#include "listaEN.h"

int menu();

int main() {
    ord::tLista lista;
    bool ok;
    ord::cargar(lista, ok);
    if (!ok) {
        cout << "No se pudo abrir el archivo!" << endl;
    }
    else {
        ord::mostrar(lista);
        ...
    }
}
```

O usar una instrucción `using namespace ord;`



Ejemplo de espacio de nombres

Uso del espacio de nombres

```
#include <iostream>
using namespace std;
#include "registrofin.h"
#include "listaEN.h"
using namespace ord;

int menu();

int main() {
    tLista lista;
    bool ok;
    cargar(lista, ok);
    if (!ok) {
        cout << "No se pudo abrir el archivo!" << endl;
    }
    else {
        mostrar(lista);
        ...
    }
}
```



Espacios de nombres

Implementaciones alternativas

Distintos espacios de nombres para distintas implementaciones

¿Lista ordenada o lista desordenada?

```
namespace ord { // Lista ordenada
    const int N = 100;
    typedef tRegistro tArray[N];
    ...
    void mostrar(const tLista &lista);
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    ...
} // namespace

namespace des { // Lista desordenada
    const int N = 100;
    typedef tRegistro tArray[N];
    ...
    void mostrar(const tLista &lista);
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    ...
} // namespace
```



Implementaciones alternativas

Todo lo común puede estar fuera de la estructura `namespace`:

```
#ifndef listaEN_H
#define listaEN_H

#include "registrofin.h"

const int N = 100;

typedef tRegistro tArray[N];
typedef struct {
    tArray registros;
    int cont;
} tLista;

void mostrar(const tLista &lista);
void eliminar(tLista &lista, int pos, bool &ok); // pos = 1..N

...
```



Implementaciones alternativas

```
namespace ord { // Lista ordenada
    const string BD = "bd.txt";
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    int buscar(tLista lista, string nombre);
    void cargar(tLista &lista, bool &ok);
    void guardar(tLista lista);
} // namespace

namespace des { // Lista desordenada
    const string BD = "bddes.txt";
    void insertar(tLista &lista, tRegistro registro, bool &ok);
    int buscar(tLista lista, string nombre);
    void cargar(tLista &lista, bool &ok);
    void guardar(tLista lista);
} // namespace

#endif
```



cargar() y guardar() se distinguen porque usan su propia BD, pero se implementan exactamente igual



Implementaciones alternativas

listaEN.cpp

```
#include <iostream>
using namespace std;
#include <fstream>
#include "listaEN.h"

// IMPLEMENTACIÓN DE LOS SUBPROGRAMAS COMUNES
void eliminar(tLista &lista, int pos, bool &ok) { // ...
}

void mostrar(const tLista &lista) { // ...
}

// IMPLEMENTACIÓN DE LOS SUBPROGRAMAS DEL ESPACIO DE NOMBRES ord
void ord::insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        int i = 0;
        while ((i < lista.cont) && (lista.registros[i] < registro)) {
            i++;
        } ...
    }
}
```



Implementaciones alternativas

```
        for (int j = lista.cont; j > i; j--) {
            lista.registros[j] = lista.registros[j - 1];
        }
        lista.registros[i] = registro;
        lista.cont++;
    }

int ord::buscar(tLista lista, string nombre) {
    int ini = 0, fin = lista.cont - 1, mitad;
    bool encontrado = false;
    while ((ini <= fin) && !encontrado) {
        mitad = (ini + fin) / 2;
        if (nombre == lista.registros[mitad].nombre) {
            encontrado = true;
        }
        else if (nombre < lista.registros[mitad].nombre) {
            fin = mitad - 1;
        }
        else {
            ini = mitad + 1;
        }
    } ...
```



Implementaciones alternativas

```
if (encontrado) {  
    mitad++;  
}  
else {  
    mitad = -1;  
}  
return mitad;  
}  
  
void ord::cargar(tLista &lista, bool &ok) { // ...  
}  
  
void ord::guardar(tLista lista) { // ...  
}  
...
```



Implementaciones alternativas

```
// IMPLEMENTACIÓN DE LOS SUBPROGRAMAS DEL ESPACIO DE NOMBRES des

void des::insertar(tLista &lista, tRegistro registro, bool &ok) {
    ok = true;
    if (lista.cont == N) {
        ok = false; // Lista llena
    }
    else {
        lista.registros[lista.cont] = registro;
        lista.cont++;
    }
}

int des::buscar(tLista lista, string nombre) {
    int pos = 0;
    bool encontrado = false;
    while ((pos < lista.cont) && !encontrado) {
        if (nombre == lista.registros[pos].nombre) {
            encontrado = true;
        }
        else {
            pos++;
        }
    } ...
```



Implementaciones alternativas

```
if (encontrado) {  
    pos++;  
}  
else {  
    pos = -1;  
}  
return pos;  
}  
  
void des::cargar(tLista &lista, bool &ok) { // ...  
}  
  
void des::guardar(tLista lista) { // ...  
}
```



Programa principal

```
#include <iostream>
using namespace std;
#include "registrofin.h"
#include "listaEN.h"
using namespace ord;

int menu();

int main() {
    tLista lista;
    bool ok;
    ...
    tRegistro registro = nuevo();
    insertar(lista, registro, ok);
    if (!ok) {
        ...
    }
}
```

1:	12345	Alvarez	120000.00
2:	11111	Benitez	100000.00
3:	21112	Dominguez	90000.00
4:	11111	Duran	120000.00
5:	22222	Fernandez	120000.00
6:	12345	Gomez	100000.00
7:	10000	Hernandez	150000.00
8:	21112	Jimenez	100000.00
9:	54321	Manzano	95000.00
10:	11111	Perez	90000.00
11:	12345	Sanchez	90000.00
12:	10000	Sergei	100000.00
13:	33333	Tarazona	120000.00
14:	12345	Turegano	100000.00
15:	11111	Urpiano	90000.00

```
1 - Insertar
2 - Eliminar
3 - Buscar
0 - Salir
Opción: 1
```

```
Introduce el código: 33333
```

```
Introduce el nombre: Calvo
```

```
Introduce el sueldo: 95000
```

1:	12345	Alvarez	120000.00
2:	11111	Benitez	100000.00
3:	33333	Calvo	95000.00
4:	21112	Dominguez	90000.00
5:	11111	Duran	120000.00
6:	22222	Fernandez	120000.00
7:	12345	Gomez	100000.00
8:	10000	Hernandez	150000.00
9:	21112	Jimenez	100000.00
10:	54321	Manzano	95000.00
11:	11111	Perez	90000.00
12:	12345	Sanchez	90000.00
13:	10000	Sergei	100000.00
14:	33333	Tarazona	120000.00
15:	12345	Turegano	100000.00
16:	11111	Urpiano	90000.00



Programa principal

```
#include <iostream>
using namespace std;
#include "registrofin.h"
#include "listaEN.h"
using namespace des;

int menu();

int main() {
    tLista lista;
    bool ok;
    ...
    tRegistro registro = nuevo();
    insertar(lista, registro, ok);
    if (!ok) {
        ...
    }
}
```

1:	12345	Alvarez	120000.00
2:	11111	Benitez	100000.00
3:	21112	Dominguez	90000.00
4:	11111	Duran	120000.00
5:	22222	Fernandez	120000.00
6:	12345	Gomez	100000.00
7:	10000	Hernandez	150000.00
8:	21112	Jimenez	100000.00
9:	54321	Manzano	95000.00
10:	11111	Perez	90000.00
11:	12345	Sanchez	90000.00
12:	10000	Sergei	100000.00
13:	33333	Tarazona	120000.00
14:	12345	Turegano	100000.00
15:	11111	Urpiano	90000.00

1 - Insertar
2 - Eliminar
3 - Buscar
0 - Salir
Opción: 1

Introduce el código: 33333
Introduce el nombre: Calvo
Introduce el sueldo: 95000

1:	12345	Alvarez	120000.00
2:	11111	Benitez	100000.00
3:	21112	Dominguez	90000.00
4:	11111	Duran	120000.00
5:	22222	Fernandez	120000.00
6:	12345	Gomez	100000.00
7:	10000	Hernandez	150000.00
8:	21112	Jimenez	100000.00
9:	54321	Manzano	95000.00
10:	11111	Perez	90000.00
11:	12345	Sanchez	90000.00
12:	10000	Sergei	100000.00
13:	33333	Tarazona	120000.00
14:	12345	Turegano	100000.00
15:	11111	Urpiano	90000.00
16:	33333	Calvo	95000.00



Calidad y reutilización del software



Software de calidad

El software debe ser desarrollado con buenas prácticas de ingeniería del software que aseguren un buen nivel de calidad

Los distintos módulos de la aplicación deben ser probados exhaustivamente, tanto de forma independiente como en su relación con los demás módulos

La prueba y depuración es muy importante y todos los equipos deberán seguir buenas pautas para asegurar la calidad

Los módulos deben ser igualmente bien documentados, de forma que otros desarrolladores puedan aprovecharlos



Prueba y depuración del software

Prueba exhaustiva

El software debe ser probado exhaustivamente

Debemos intentar descubrir todos los errores posibles

Los errores deben ser depurados, corrigiendo el código

Pruebas sobre listas:

- ✓ Lista inicialmente vacía
- ✓ Lista inicialmente llena
- ✓ Lista con un número intermedio de elementos
- ✓ Archivo no existente

Etcétera...

Se han de probar todas las opciones/situaciones del programa

En las clases prácticas veremos cómo se depura el software



Reutilización del software

No reinventemos la rueda

Desarrollar el software pensando en su posible reutilización

Un software de calidad debe poder ser fácilmente reutilizado

Nuestros módulos deben ser fácilmente usados y modificados

Por ejemplo: Nueva aplicación que gestione una lista de longitud variable de registros con NIF, nombre, apellidos y edad

Partiremos de los módulos **registro** y **lista** existentes

Las modificaciones básicamente afectarán al módulo **registro**



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.

