

8

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	2
Interfaz frente a implementación	7
Uso de módulos de biblioteca	13
Ejemplo: Gestión de una tabla de datos ordenada I	12
Compilación de programas multiarchivo	22
El preprocesador	24
Cada cosa en su módulo	26
Ejemplo: Gestión de una tabla de datos ordenada II	27
El problema de las inclusiones múltiples	33
Compilación condicional	38
Protección frente a inclusiones múltiples	39
Ejemplo: Gestión de una tabla de datos ordenada III	40
Implementaciones alternativas	48
Espacios de nombres	52
Implementaciones alternativas	61
Calidad y reutilización del software	72



Fundamentos de la programación

Programas multiarchivo y compilación separada

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 2

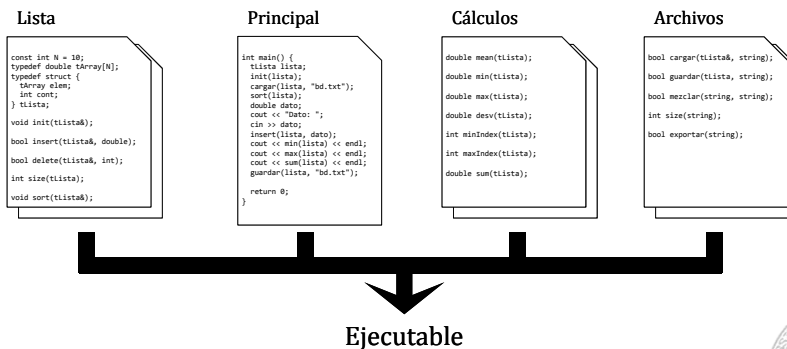


Programación modular

Programas multiarchivo

El código fuente del programa se reparte entre varios archivos (*módulos*), cada uno con las declaraciones y los subprogramas que tienen relación.

→ Módulos: archivos de código fuente con declaraciones y subprogramas de una unidad funcional: una estructura de datos, un conjunto de utilidades, ...



Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

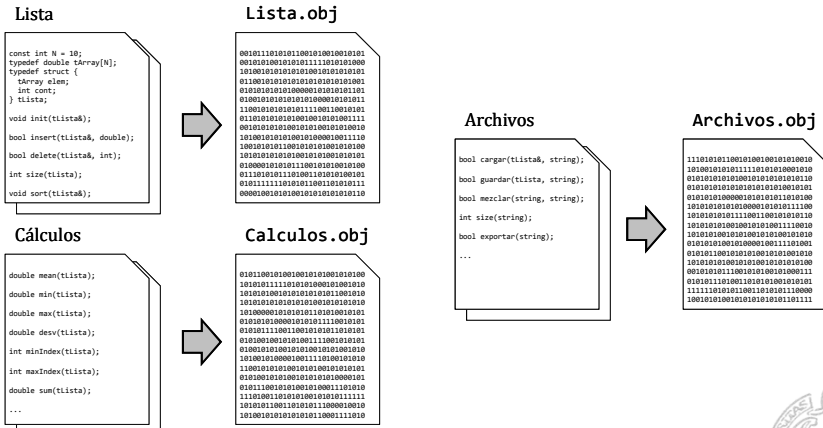
Página 3



Programación modular

Compilación separada

Cada módulo se compila a código fuente de forma independiente:



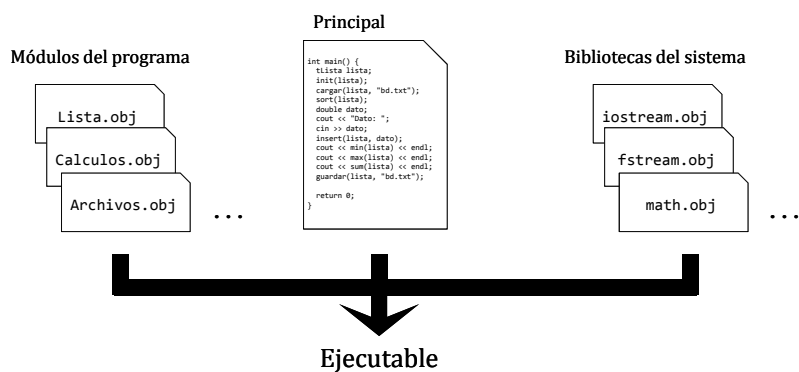
Luis Hernández Yáñez



Programación modular

Compilación separada

Al compilar el programa principal, se adjuntan los módulos compilados:



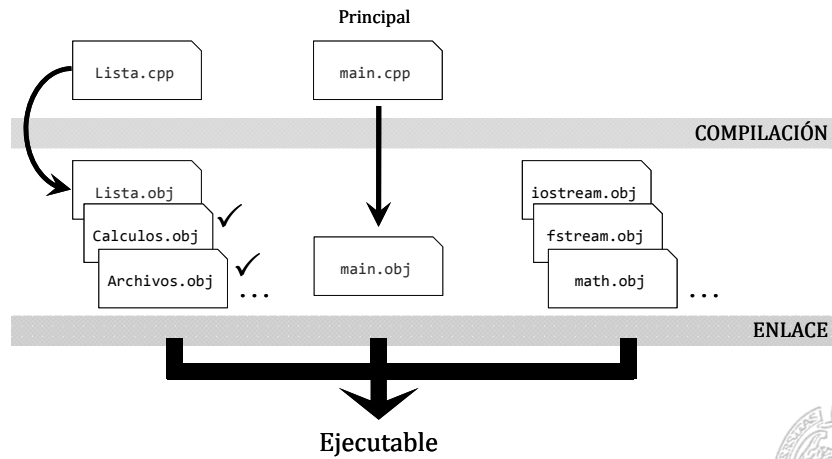
Luis Hernández Yáñez



Programación modular

Compilación separada

¡Sólo los archivos de código fuente modificados necesitan ser recompilados!



Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 6



Fundamentos de la programación

Interfaz frente a implementación

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 7



Interfaz frente a implementación

Creación de módulos de biblioteca

En el código de un programa de un único archivo tenemos:

- ✓ Definiciones de constantes.
- ✓ Declaraciones de tipos de datos.
- ✓ [Quizás, variables globales. *Mejor evitar tener variables globales.*]
- ✓ Prototipos de los subprogramas.
- ✓ Implementación de los subprogramas.
- ✓ Implementación de la función `main()`.

Las constantes, tipos [, variables] y prototipos de subprogramas que tienen que ver con alguna unidad funcional indican *cómo se usa* ésta: interfaz.

- ✓ Estructura de datos con los subprogramas que la gestionan.
- ✓ Conjunto de utilidades (subprogramas) de uso general.
- ✓ Etcétera.

La implementación de los subprogramas es eso, implementación.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 8



Interfaz frente a implementación

Creación de módulos de biblioteca

Interfaz: Definiciones y declaraciones de datos y subprogramas (prototipos).

¡Es todo lo que el usuario de esa unidad funcional necesita saber!

Implementación: Código de los subprogramas que hacen el trabajo.

No necesita conocerse para utilizarlo: ¡Se da por sentado que es correcto!

Separamos la interfaz y la implementación en dos archivos separados:

- ✓ Archivo de cabecera: Definiciones y declaraciones de datos y subprogramas.
- ✓ Archivo de implementación: Implementación de los subprogramas.

Archivos de cabecera: extensión `.h`

Archivos de implementación: extensión `.cpp` } **Mismo nombre (x.h / x.cpp)**

Extraemos las definiciones y declaraciones de datos y subprogramas de la unidad funcional y las colocamos en un archivo de cabecera.

Extraemos las implementaciones de esos subprogramas y los colocamos en el archivo de implementación de ese archivo de cabecera.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 9



Interfaz frente a implementación

Creación de módulos de biblioteca

Interfaz frente a implementación

Lista.h	Lista.cpp	Módulo Unidad Biblioteca
<pre>const int N = 10; typedef double tArray[N]; typedef struct { tArray elem; int cont; } tLista; void init(tLista&); bool insert(tLista&, double); bool delete(tLista&, int); int size(tLista); void sort(tLista&);</pre>	<pre>#include "Lista.h" void init(tLista& lista) { lista.cont = 0; } bool insert(tLista& lista, double valor) { if (lista.cont == N) return false; else { lista.elem[lista.cont] = valor; lista.cont++; } }</pre>	

Si otro módulo (o el programa principal) quiere usar algo de esa biblioteca: Deberá incluir el archivo de cabecera.

main.cpp

```
#include "Lista.h"
...
```

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

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 10



Interfaz frente a implementación

Creación de módulos de biblioteca

Interfaz

El archivo de cabecera (.h) tiene todo lo que necesita conocer cualquier otro módulo (o programa principal) que quiera utilizar los servicios (subprogramas) de ese módulo de biblioteca.

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

main.cpp

```
#include "Lista.h"
...
```

Preprocesador



main.cpp

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

void init(tLista&);
bool insert(tLista&, double);
bool delete(tLista&, int);
int size(tLista);
...
```

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

Lista.h

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

void init(tLista&);
bool insert(tLista&, double);
bool delete(tLista&, int);
int size(tLista);
void sort(tLista&);
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 11



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"
...
```

Cuando se compila el módulo se genera el código objeto.

Mientras no se modifique el código fuente no hay necesidad de recompilar el módulo.

Código que usa el módulo:

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

Lista.cpp

```
#include "Lista.h"
void init(lista& lista) {
    lista.cont = 0;
}
bool insert(lista& lista,
double valor) {
    if (lista.cont == N)
        return false;
    else {
        lista.elem[lista.cont] =
        valor;
        lista.cont++;
    }
}
```

Lista.obj

```
0010110101011001010010010101
0010100010010111100101000
101000101010101010101010101
011000101010101010101010101
01010101010000010101010101
010001010101010000101010101
1100010101011100100010101
0110101010101010101010111
001010100001010001010010
1010010101010100001001110
10010101100101010101010100
1010101010101010101010101
0100001010110010101010100
01110101010001010101010101
0101111010101001010101011
000100101010101010101010
```



Fundamentos de la programación

Uso de módulos de biblioteca



Programación modular

Uso de módulos de biblioteca

Ejemplo: Gestión de una tabla de datos ordenada (Tema 7)

Todo lo que tenga que ver con la tabla en sí estará en su propio módulo.

Ahora el código estará repartido en tres archivos:

- ✓ `tabla.h`: archivo de cabecera del módulo de tabla de datos
- ✓ `tabla.cpp`: archivo de implementación del módulo de tabla de datos
- ✓ `bd.cpp`: programa principal que usa el módulo de tabla de datos

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

Como es un módulo propio de la aplicación, en la directiva `#include` usamos dobles comillas:

```
#include "tabla.h"
```

Los archivos de cabecera de las bibliotecas del sistema siempre se encierran entre ángulos y no tienen necesariamente que llevar extensión `.h`.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 14



Programación modular

Archivo de cabecera

Módulo: Gestión de una tabla de datos ordenada I

`tabla.h`

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

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

¡Documenta bien el código!

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 15



Programación modular

Archivo de cabecera

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

Cada prototipo irá con un comentario que explique la utilidad del subprograma, los datos de E/S, particularidades, ... (Aquí se omiten por cuestión de espacio.)

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 16



Programación modular

Archivo de implementación

Módulo: Gestión de una tabla de datos ordenada I

tabla.cpp

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

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

...
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 17



Programación modular

Archivo de implementación

```
bool insertar(tTabla &tabla, tRegistro registro) {
    bool ok = true;
    if (tabla.cont == N)
        ok = false; // Tabla llena
    else {
        int pos = 0;
        bool enc= false;
        //búsqueda asimétrica
        while ((pos < tabla.cont) && !enc){
            if(tabla.registros[pos] > registro) enc= true;
            else pos++;
        }

        for (int j = tabla.cont; j > pos; j--)
            // Desplazamos una posición a la derecha
            tabla.registros[j] = tabla.registros[j-1];
        tabla.registros[pos] = registro;
        tabla.cont++;
    }
    return ok;
}
...
```

Luis Hernández Yáñez
Pedro J. Martín de la Calle



Fundamentos de la programación: Programación modular

Página 18



Programación modular

Archivo de implementación

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

int buscar(tTabla tabla, string nombre) {
    // Devuelve -1 si no se ha encontrado
    int ini = 0, fin = tabla.cont - 1, mitad;
    bool encontrado = false;
    while ((ini <= fin) && !encontrado) {
        ...
    }
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 19



Programación modular

Programa principal

Módulo: Gestión de una tabla de datos ordenada I

bd.cpp

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

int menu();

int menu() {
    cout << endl;
    cout << "1 - Insertar" << endl;
    cout << "2 - Eliminar" << endl;
    cout << "3 - Buscar" << endl;
    cout << "0 - Salir" << endl;
    int op;
    do {
        cout << "Elige: ";
        cin >> op;
    } while ((op < 0) || (op > 3));
    return op;
}
...
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 20



Programación modular

Programa principal

```
int main() {
    tTabla tabla;
    if (!cargar(tabla))
        cout << "Error al abrir el archivo!" << endl;
    else {
        mostrar(tabla);
        int op;
        do {
            op = menu();
            if (op == 1) {
                tRegistro registro = nuevo();
                if (!insertar(tabla, registro))
                    cout << "Error: tabla llena!" << endl;
                else
                    mostrar(tabla);
            }
            else if (op == 2) {
                int pos;
                cout << "Posicion: ";
                cin >> pos;
                if (!eliminar(tabla, pos))
                    cout << "Error: Posicion inexistente!" << endl;
                ...
            }
        }
    }
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 21



Fundamentos de la programación

Compilación de programas multiarchivo

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 22



Programación modular

Compilación de programas multiarchivo

G++

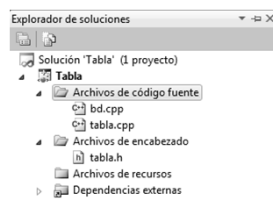
Si están todos los archivos de cabecera y de implementación en el mismo directorio simplemente listamos todos los .cpp en la orden g++:

```
D:\FP\Tema9>g++ -o bd.exe tabla.cpp bd.cpp
```

Recuerda que sólo se compilan los .cpp.

Visual C++/Studio

Muestra los archivos de cabecera y de implementación internamente organizados en grupos distintos:



A los archivos de cabecera los llama de encabezado.

Con la opción Generar solución se compilan todos los .cpp.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 23



Fundamentos de la programación

El preprocesador

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 24



Programación modular

El Preprocesador

Directivas: # . . .

Antes de realizar la compilación se pone en marcha el *preprocesador*.

Interpreta las directivas y genera un único archivo temporal con todo el código del módulo o programa principal. Como en la inclusión (directiva `#include`):

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

const int N = 100;

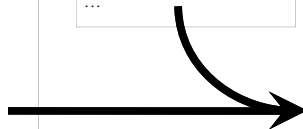
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

typedef tRegistro tLista[N];

typedef struct {
    tLista registros;
    int cont;
} tTabla;

...
```

```
#include "tabla.h"
int menu();
...
```



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

const int N = 100;

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

typedef tRegistro tLista[N];

typedef struct {
    tLista registros;
    int cont;
} tTabla;

...

int menu();
...
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 25



Fundamentos de la programación

Cada cosa en su módulo

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 26



Programación modular

Distribuir la funcionalidad del programa en módulos

Un módulo encapsula un conjunto de subprogramas que tienen relación entre sí.

La relación puede venir dada por una estructura de datos sobre la que trabajan. O por ser todos subprogramas de un determinado contexto de aplicación (bibliotecas de funciones matemáticas, de fecha, etcétera).

A menudo las estructuras de datos contienen otras estructuras:

```
const int N = 100;
typedef struct {
    int codigo;
    string nombre;
    double sueldo;
} tRegistro;

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

Una tabla es una lista de registros.

- ✓ Estructura tRegistro
- ✓ Estructura tTabla
(contiene datos de tipo tRegistro)

La gestión de cada estructura, en su módulo.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 27



Programación modular

Archivo de cabecera

Gestión de una tabla de datos ordenada II

registro.h

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

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

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

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 28



Programación modular

Archivo de implementación

Gestión de una tabla de datos ordenada II

registro.cpp

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

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

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

...
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 29



Programación modular

Archivo de cabecera

Gestión de una tabla de datos ordenada II

tabla2.h

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

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

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

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 30



Programación modular

Archivo de implementación

Gestión de una tabla de datos ordenada II

tabla2.cpp

```
#include <iostream>
#include <fstream>
using namespace std;
#include "tabla2.h" ←

bool insertar(tTabla& tabla, tRegistro registro) {
    bool ok = true;
    if (tabla.cont == N)
        ok = false; // tabla llena
    else {
        int pos = 0;
        bool enc = false;
        //búsqueda asimétrica
        while ((pos < tabla.cont) && !enc){
            if(tabla.registros[pos] > registro) enc = true;
            else pos++;
        }
        for (int j = tabla.cont; j > pos; j--)
            tabla.registros[j] = tabla.registros[j-1];
        tabla.registros[pos] = registro;
        tabla.cont++;
    }
    return ok;
}
...
```

Luis Hernández Yáñez
Pedro J. Martín de la Calle



Fundamentos de la programación: Programación modular

Página 31



Programación modular

Programa principal

Gestión de una tabla de datos ordenada II

bd2.cpp

```

#include <iostream>
using namespace std;
#include "registro.h" ←
#include "tabla2.h" ←

int menu();

int menu() {
    cout << endl;
    cout << "1 - Insertar" << endl;
    cout << "2 - Eliminar" << endl;
    cout << "3 - Buscar" << endl;
    cout << "0 - Salir" << endl;
    int op;
    do {
        cout << "Elige: ";
        cin >> op;
    } while ((op < 0) || (op > 3));
    return op;
}
...

```

No intentes compilar este ejemplo.

Tiene errores...

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 32



Fundamentos de la programación

El problema de las inclusiones múltiples

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

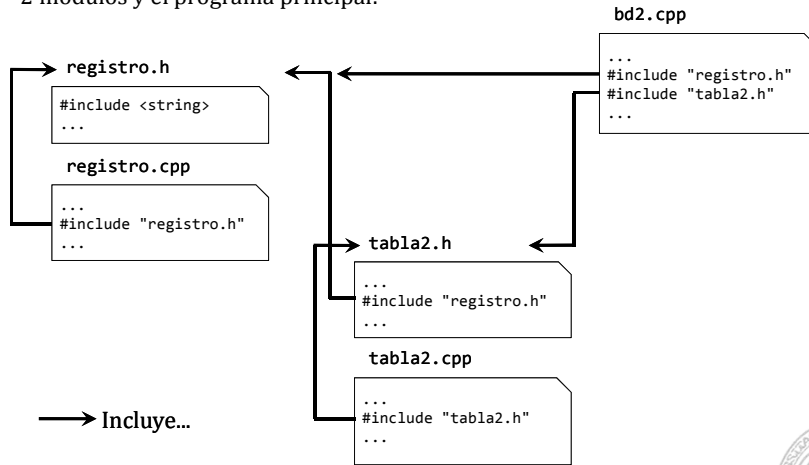
Página 33



Programación modular

Gestión de una tabla de datos ordenada II

2 módulos y el programa principal:



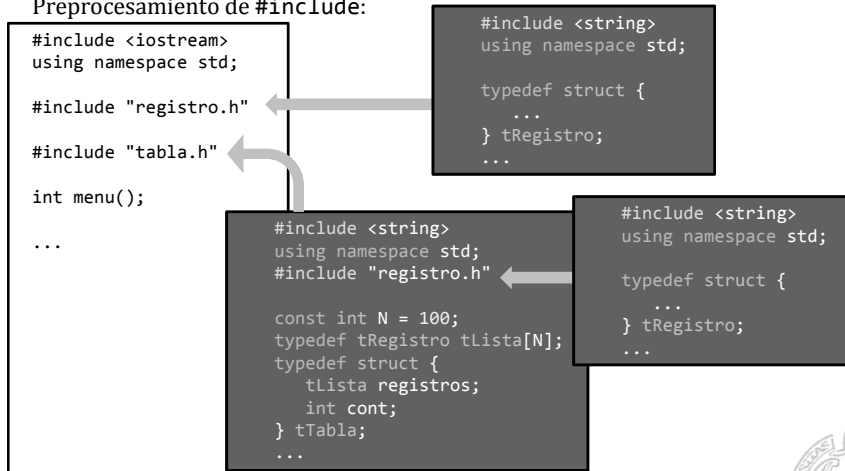
Luis Hernández Yáñez



Programación modular

Gestión de una tabla de datos ordenada II

Preprocesamiento de #include:



Luis Hernández Yáñez



Programación modular

Gestión de una tabla de datos ordenada II

Preprocesamiento de #include:

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

#include <string>
using namespace std;

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

#include "tabla.h"

int menu();

...
```

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

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

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

 Sustituido

Luis Hernández Yáñez



Programación modular

Gestión de una tabla de datos ordenada II

Preprocesamiento de #include:

```
#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 tLista[N];
typedef struct {
    tLista registros;
    int cont;
} tTabla;
...

int menu();

...
```

¡Identificador duplicado!

Luis Hernández Yáñez



Programación modular

Compilación condicional

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

Se usan en conjunción con la directiva `#define`.

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

La directiva `#define`, como su nombre indica, define un símbolo (identificador).

En el caso de la izquierda se compilará el "Código if" y no el "Código else", en caso de que lo haya. En el caso de la derecha, al revés, o nada si no hay else.

Las cláusulas else son opcionales.

Directivas útiles para configurar distintas versiones.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 38



Programación modular

Protección frente a inclusiones múltiples

`tabla.cpp` incluye `registro.h` y `bd2.cpp` también incluye `registro.h`
 → ¡Identificadores duplicados!

Cada módulo debe incluirse una y sólo una vez.

Protección frente a inclusiones múltiples:

```
#ifndef X
#define X
... // Código del módulo
#endif
```

El símbolo *X* debe ser único
para cada módulo.

La primera vez que se encuentra ese código el preprocesador, incluye el código del módulo por no estar definido el símbolo *X*, pero al mismo tiempo define ese símbolo. De esta forma, la siguiente vez que se encuentre ese `#ifndef` ya no vuelve a incluir el código del módulo, pues ya ha sido definido el símbolo *X*.

Para cada módulo elegimos como símbolo *X* el nombre del archivo, sustituyendo el punto por un subrayado: `REGISTRO_H`, `TABLA_H`, ...

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 39



Programación modular

Archivo de cabecera

Gestión de una tabla de datos ordenada III

registrofin.h

```
#ifndef REGISTROFIN_H
#define REGISTROFIN_H

#include <string>
using namespace std;

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

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

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 40



Programación modular

Archivo de implementación

Gestión de una tabla de datos ordenada III

registrofin.cpp

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

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

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

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 41



Programación modular

Archivo de cabecera

Gestión de una tabla de datos ordenada III

tablafin.h

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

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

void mostrar(const tTabla &tabla);
bool insertar(tTabla &tabla, tRegistro registro);
bool eliminar(tTabla &tabla, int pos); // pos = 1..N
int buscar(tTabla tabla, string nombre);
bool cargar(tTabla &tabla);
void guardar(tTabla tabla);
#endif
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 42



Programación modular

Archivo de implementación

Gestión de una tabla de datos ordenada III

tablafin.cpp

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

bool insertar(tTabla &tabla, tRegistro registro) {
    bool ok = true;
    if (tabla.cont == N)
        ok = false; // tabla llena
    else {
        int pos = 0;
        bool enc = false;
        //búsqueda asimétrica
        while ((pos < tabla.cont) && !enc){
            if(tabla.registros[pos] > registro) enc = true;
            else pos++;
        }

        for (int j = tabla.cont; j > pos; j--)
            tabla.registros[j] = tabla.registros[j-1];

        tabla.registros[pos] = registro;
        tabla.cont++;
    }
    return ok;
} ...
```

Luis Hernández Yáñez
Pedro J. Martín de la Calle



Fundamentos de la programación: Programación modular

Página 43



Programación modular

Gestión de una tabla de datos 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 "tablafin.h" ←
```

```
int menu();
```

```
...
```

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

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

TABLAFIN_H no se ha definido todavía

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 46



Programación modular

Gestión de una tabla de datos 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 TABLAFIN_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 se ha definido!*

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 47



Fundamentos de la programación

Implementaciones alternativas

Luis Hernández Yáñez



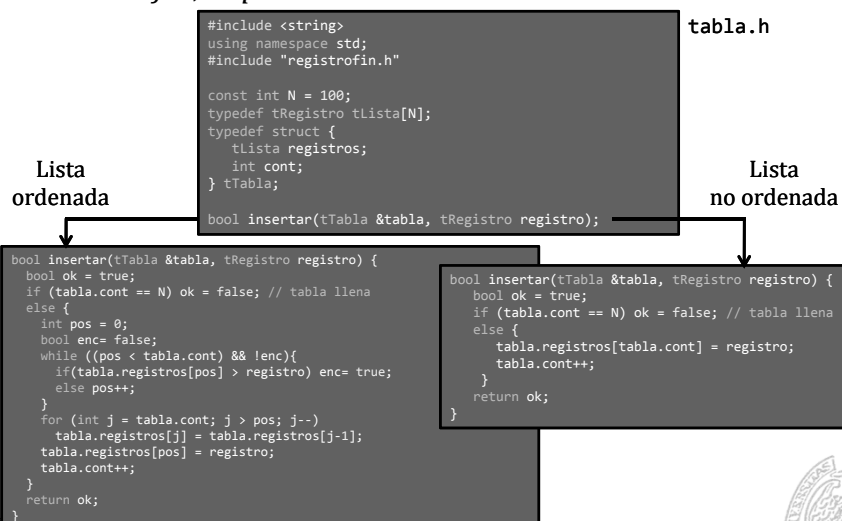
Fundamentos de la programación: Programación modular

Página 48



Implementaciones alternativas

Misma interfaz, implementación alternativa

Luis Hernández Yáñez
Pedro J. Martín de la Calle

Fundamentos de la programación: Programación modular

Página 49



Implementaciones alternativas

Misma interfaz, implementación alternativa

tablaORD.cpp: Lista ordenada

```
...
#include "tabla.h"

bool insertar(tTabla &tabla, tRegistro registro) {
    bool ok = true;
    if (tabla.cont == N) ok = false; // tabla llena
    else {
        int pos = 0;
        bool enc = false;
        while ((pos < tabla.cont) && !enc){
            if(tabla.registros[pos] > registro) enc= true;
            else pos++;
        }
        for (int j = tabla.cont; j > pos; j--){
            tabla.registros[j] = tabla.registros[j-1];
            tabla.registros[pos] = registro;
            tabla.cont++;
        }
        return ok;
    }
    ...
}
```

tablaDES.cpp: Lista no ordenada

```
...
#include "tabla.h"

bool insertar(tTabla &tabla, tRegistro registro) {
    bool ok = true;
    if (tabla.cont == N) ok = false; // tabla llena
    else {
        tabla.registros[tabla.cont] = registro;
        tabla.cont++;
    }
    return ok;
}
...
}
```

Luis Hernández Yáñez
Pedro J. Martín de la Calle



Fundamentos de la programación: Programación modular

Página 50



Implementaciones alternativas

Misma interfaz, implementación alternativa

Al compilar, incluimos un archivo de implementación u otro:

¿Programa con tabla ordenada o con tabla desordenada?

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

Incluye la implementación de la tabla con ordenación.

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

Incluye la implementación de la tabla sin ordenación.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 51



Fundamentos de la programación

Espacios de nombres

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 52



Espacios de nombres

Agrupaciones lógicas de declaraciones

Un *espacio de nombres* permite agrupar entidades (tipos, variables, funciones) bajo un nombre distintivo. Se puede considerar que el ámbito global del programa completo está dividido en subámbitos, cada uno con su propio nombre.

Forma de un espacio de nombres:

```
namespace nombre {
    // Declaraciones (entidades)
}
```

Por ejemplo:

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

Se declaran las variables *i* y *d* dentro del espacio de nombres *miEspacio*.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 53



Espacios de nombres

Acceso a miembros de un espacio de nombres

Para acceder a las entidades declaradas dentro de un espacio de nombres hay que utilizar el *operador de resolución de ámbito (::)*.

Por ejemplo, para acceder a las variables declaradas en el espacio de nombres `miEspacio` cualificamos esos identificadores con el nombre del espacio y el operador de resolución de ámbito:

```
miEspacio::i
miEspacio::d
```

Los espacios de nombres resultan útiles en aquellos casos en los que pueda haber entidades con el mismo identificador en distintos módulos o en ámbitos distintos de un mismo módulo.

Encerraremos 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`.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 54



Espacios de nombres

using

La palabra clave `using` se utiliza para introducir un elemento de un espacio de nombres en el ámbito de declaraciones 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 de nombres explícito
    cout << segundo::x << endl; // espacio de nombres explícito
    return 0;
}
```

```
5
2.7183
10
3.1416
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 55



Espacios de nombres

using namespace

La instrucción `using namespace` se utiliza para introducir todos los elementos de un espacio de nombres en el ámbito de declaraciones actual:

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

```
namespace primero {
    int x = 5;
    int y = 10;
}
namespace segundo {
    double x = 3.1416;
    double y = 2.7183;
}
```

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

```
5
10
3.1416
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 de nombres explícito
    cout << segundo::y << endl; // espacio de nombres explícito
    return 0;
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 56



Espacios de nombres

Ejemplo de uso de espacios de nombres

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

namespace ord { // Tabla ordenada
    const int N = 100;
    typedef tRegistro tLista[N];
    typedef struct {
        tLista registros;
        int cont;
    } tTabla;
    const char BD[] = "bd.txt";

    void mostrar(const tTabla &tabla);
    bool insertar(tTabla &tabla, tRegistro registro);
    bool eliminar(tTabla &tabla, int pos); // pos = 1..N
    int buscar(tTabla tabla, string nombre);
    bool cargar(tTabla &tabla);
    void guardar(tTabla tabla);
} // namespace

#endif
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 57



Espacios de nombres

Archivo de implementación

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

// IMPLEMENTACIÓN DE LAS FUNCIONES DEL ESPACIO DE NOMBRES ord

bool ord::insertar(tTabla &tabla, tRegistro registro) { // ...
}

bool ord::eliminar(tTabla &tabla, int pos) { // ...
}

int ord::buscar(tTabla tabla, string nombre) { // ...
}

void ord::mostrar(const tTabla &tabla) { // ...
}

bool ord::cargar(tTabla &tabla) { // ...
}

void ord::guardar(tTabla tabla) { // ...
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 58



Espacios de nombres

Uso del espacio de nombres

Cualquier módulo que utilice `tablaEN.h` debe cualificar cada nombre de esa biblioteca con el nombre del espacio de nombres en que se encuentra:

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

int menu();

int main() {
    ord::tTabla tabla;

    if (!ord::cargar(tabla)) {
        cout << "Error al abrir el archivo!" << endl;
    }
    else {
        ord::mostrar(tabla);
        ...
    }
}
```

O utilizar una instrucción `using namespace ord;`

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 59



Espacios de nombres

Uso de espacios de nombres

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

int menu();

int main() {
    tTabla tabla;
    if (!cargar(tabla)) {
        cout << "Error al abrir el archivo!" << endl;
    }
    else {
        mostrar(tabla);
        ...
    }
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 60



Espacios de nombres

Implementaciones alternativas

Distintos espacios de nombres para distintas implementaciones.

¿Tabla ordenada o tabla desordenada?

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

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

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 61



Espacios de nombres

Archivo de cabecera

Implementaciones alternativas

tablaEN.h

Todo lo que sea común puede estar fuera de la estructura namespace:

```
#ifndef TABLAEN_H
#define TABLAEN_H

#include "registrofin.h"

const int N = 100;

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

void mostrar(const tTabla &tabla);
bool eliminar(tTabla& tabla, int pos); // pos = 1..N

...
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 62



Espacios de nombres


Implementaciones alternativas

```
namespace ord { // Tabla ordenada
    const char BD[] = "bd.txt";
    bool insertar(tTabla &tabla, tRegistro registro);
    int buscar(tTabla tabla, string nombre);
    bool cargar(tTabla &tabla);
    void guardar(tTabla tabla);
} // namespace

namespace des { // Tabla desordenada
    const char BD[] = "bddes.txt";

    bool insertar(tTabla &tabla, tRegistro registro);
    int buscar(tTabla tabla, string nombre);
    bool cargar(tTabla &tabla);
    void guardar(tTabla tabla);
} // namespace

#endif
```

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

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 63



Espacios de nombres

Archivo de implementación

Implementaciones alternativas

tablaEN.cpp

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

bool eliminar(tTabla &tabla, int pos) { // ...
}
void mostrar(const tTabla &tabla) { // ...
}

// IMPLEMENTACIÓN DE LAS FUNCIONES DEL ESPACIO DE NOMBRES ord
bool ord::insertar(tTabla& tabla, tRegistro registro) {
    bool ok = true;
    if (tabla.cont == N) ok = false;
    else {
        int pos = 0;
        bool enc = false;
        while ((pos < tabla.cont) && !enc){
            if(tabla.registros[pos] > registro) enc = true;
            else pos++;
        }
        for (int j = tabla.cont; j > pos; j--)
            tabla.registros[j] = tabla.registros[j-1];
        tabla.registros[pos] = registro;
        tabla.cont++;
    }
    return ok;
} ...
```

Luis Hernández Yáñez
Pedro J. Martín de la Calle



Fundamentos de la programación: Programación modular

Página 64



Espacios de nombres

Implementaciones alternativas

```
int ord::buscar(tTabla tabla, string nombre) {
    int ini = 0, fin = tabla.cont - 1, mitad;
    bool encontrado = false;
    while ((ini <= fin) && !encontrado) {
        mitad = (ini + fin) / 2;
        if (nombre == tabla.registros[mitad].nombre) encontrado = true;
        else if (nombre < tabla.registros[mitad].nombre) fin = mitad - 1;
        else ini = mitad + 1;
    }
    if (encontrado) mitad++;
    else mitad = -1;
    return mitad;
}
bool ord::cargar(tTabla &tabla) {
    bool ok = true;
    ifstream archivo;
    archivo.open(BD);
    if (!archivo.is_open()) ok = false;
    else {
        tabla.cont = 0;
        int cod;
        string nom;
        double suel;
        ...
    }
}
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 65



Espacios de nombres

Implementaciones alternativas

```

while ((tabla.cont < N) && (archivo >> cod) && (archivo >> nom)
&& (archivo >> suel)) {
    tRegistro registro;
    registro.codigo = cod;
    registro.nombre = nom;
    registro.sueldo = suel;
    tabla.registros[tabla.cont] = registro;
    tabla.cont++;
}
}
return ok;
}

void ord::guardar(tTabla tabla) {
    ofstream salida;
    salida.open(BD);
    for (int i = 0; i < tabla.cont; i++)
        salida << tabla.registros[i].codigo << " "
        << tabla.registros[i].nombre << " "
        << tabla.registros[i].sueldo << endl;
    salida.close();
}
...

```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 66



Espacios de nombres

Implementaciones alternativas

```

// IMPLEMENTACIÓN DE LAS FUNCIONES DEL ESPACIO DE NOMBRES des
bool des::insertar(tTabla &tabla, tRegistro registro) {
    bool ok = true;
    if (tabla.cont == N) ok = false;
    else {
        tabla.registros[tabla.cont] = registro;
        tabla.cont++;
    }
    return ok;
}

int des::buscar(tTabla tabla, string nombre) {
    int pos = 0;
    bool encontrado = false;
    while ((pos < tabla.cont) && !encontrado) {
        if (nombre == tabla.registros[pos].nombre) encontrado = true;
        else pos++;
    }
    if (encontrado) pos++;
    else pos = -1;
    return pos;
}
...

```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 67



Espacios de nombres

Implementaciones alternativas

```
bool des::cargar(tTabla &tabla) {
    bool ok = true;
    ifstream archivo;
    archivo.open(BD);
    if (!archivo.is_open()) ok = false;
    else {
        tabla.cont = 0;
        int cod;
        string nom;
        double suel;
        while ((tabla.cont < N) && (archivo >> cod) && (archivo >> nom)
            && (archivo >> suel)) {
            tRegistro registro;
            registro.codigo = cod;
            registro.nombre = nom;
            registro.sueldo = suel;
            tabla.registros[tabla.cont] = registro;
            tabla.cont++;
        }
    }
    return ok;
}
...

```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 68



Espacios de nombres

Implementaciones alternativas

```
void des::guardar(tTabla tabla) {
    ofstream salida;
    salida.open(BD);
    for (int i = 0; i < tabla.cont; i++)
        salida << tabla.registros[i].codigo << " "
            << tabla.registros[i].nombre << " "
            << tabla.registros[i].sueldo << endl;
    salida.close();
}

```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 69



Espacios de nombres

bdEN.cpp

Implementaciones alternativas

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

int menu();

int main() {
    tTabla tabla;
    ...
    tRegistro registro = nuevo();
    if (!insertar(tabla, registro))
        ...
}
```

```
D:\FP\Tema9\tablaEN
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
Elige: 1
Introduce el codigo: 33333
Introduce el nombre: Calvo
Introduce el sueldo: 95000
1: 12345 Alvarez 120000.00
2: 11111 Benitez 100000.00
3: 33333 Calvo 95000.00
4: 21112 Dominguez 90000.00
5: 11111 Duran 120000.00
6: 22222 Fernandez 120000.00
7: 12345 Gomez 100000.00
8: 10000 Hernandez 150000.00
9: 21112 Jimenez 100000.00
10: 54321 Manzano 95000.00
11: 11111 Perez 90000.00
12: 12345 Sanchez 90000.00
13: 10000 Sergei 100000.00
14: 33333 Tarazona 120000.00
15: 12345 Turegano 100000.00
16: 11111 Urpiano 90000.00
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 70



Espacios de nombres

bdEN.cpp

Implementaciones alternativas

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

int menu();

int main() {
    tTabla tabla;
    ...
    tRegistro registro = nuevo();
    if (!insertar(tabla, registro))
        ...
}
```

```
D:\FP\Tema9\tablaEN
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
Elige: 1
Introduce el codigo: 33333
Introduce el nombre: Calvo
Introduce el sueldo: 95000
1: 12345 Alvarez 120000.00
2: 11111 Benitez 100000.00
3: 21112 Dominguez 90000.00
4: 11111 Duran 120000.00
5: 22222 Fernandez 120000.00
6: 12345 Gomez 100000.00
7: 10000 Hernandez 150000.00
8: 21112 Jimenez 100000.00
9: 54321 Manzano 95000.00
10: 11111 Perez 90000.00
11: 12345 Sanchez 90000.00
12: 10000 Sergei 100000.00
13: 33333 Tarazona 120000.00
14: 12345 Turegano 100000.00
15: 11111 Urpiano 90000.00
16: 33333 Calvo 95000.00
```

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 71



Fundamentos de la programación

Calidad y reutilización del software

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 72



Calidad del software

Software de calidad

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

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

El proceso de 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 en otros proyectos.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 73



Prueba y depuración del software

Prueba exhaustiva

El software debe ser probado exhaustivamente
 Debemos intentar descubrir todos los errores posible
 Los errores deben ser depurados, corrigiendo el código
 Pruebas sobre listas:

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

Se han de probar todas las opciones/situaciones del programa
 En las clases prácticas veremos cómo se depura el software

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 74



Reutilización del software

No reinventemos la rueda

Debemos desarrollar el software pensando en su posible reutilización.
 Un software de calidad y bien documentado debe poder ser fácilmente reutilizado.

Los módulos de nuestra aplicación deben poder ser fácilmente usados, quizá con modificaciones, en otras aplicaciones con necesidades parecidas.

Por ejemplo, si necesitamos una aplicación que gestione una lista de longitud variable de registros con NIF, nombre, apellidos y edad, partiríamos de los módulos `registro` y `tabla` la aplicación anterior. Las modificaciones básicamente afectarían al módulo `registro`.

Luis Hernández Yáñez



Fundamentos de la programación: Programación modular

Página 75



Referencias bibliográficas



- ✓ *El lenguaje de programación C++* (Edición especial)
B. Stroustrup. Addison-Wesley, 2002






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.

