Fundamentos de la programación

5 Tipos de datos estructurados

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

Luis Hernández Yáñez/Pablo Moreno Ger
Facultad de Informática
Universidad Complutense

Índice

Luis Hernández Yáñez/Pablo Moreno Ger

Tipos de datos

Luis Hernández Yáñez/Pablo Moreno Ger

Tipos de datos

Clasificación de tipos

- ✓ Simples
  - Estándar: `int`, `float`, `double`, `char`, `bool`
    Conjunto de valores predeterminado
  - Definidos por el usuario: *enumerados*
    Conjunto de valores definido por el programador
- ✓ Estructurados
  - Colecciones homogéneas: *arrays*
    Todos los elementos del mismo tipo
  - Colecciones heterogéneas: *estructuras*
    Los elementos pueden ser de tipos distintos

Tipos estructurados

Colecciones o tipos aglomerados

Agrupaciones de datos (elementos):

- ✓ Todos del mismo tipo: *array* o *tabla*
- ✓ De tipos distintos: *estructura*, *registro* o *tupla*

Arrays (tablas)

- ➢ Elementos organizados por posición: 0, 1, 2, 3, ...
- ➢ Acceso por índice: 0, 1, 2, 3, ...
- ➢ Una o varias dimensiones

Estructuras (tuplas, registros)

- ➢ Elementos (campos) sin orden establecido
- ➢ Acceso por nombre

Luis Hernández Yáñez/Pablo Moreno Ger

Arrays de nuevo

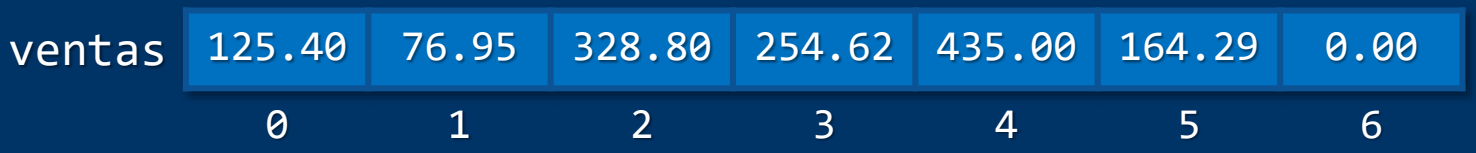
Luis Hernández Yáñez/Pablo Moreno Ger

Arrays

Estructura secuencial

Cada elemento se encuentra en una posición (*índice*):

- ✓ Los índices son enteros positivos
- ✓ El índice del primer elemento siempre es 0
- ✓ Los índices se incrementan de uno en uno



Acceso directo []

A cada elemento se accede a través de su índice:

`ventas[4]` accede al 5º elemento (contiene el valor `435.00`)

```
cout << ventas[4];
ventas[4] = 442.75;
```

Datos de un mismo tipo base:
Se usan como cualquier variable

Luis Hernández Yáñez/Pablo Moreno Ger

Tipos y variables arrays

Declaración de tipos de arrays

```
const int Dimensión = ...;
typedef tipo_base tNombre[Dimensión];
```

Ejemplo:

```
const int Dias = 7;
typedef double tVentas[Dias];
```

Declaración de variables de tipos array: como cualquier otra

```
tVentas ventas;
```

¡NO se inicializan los elementos automáticamente!

¡Es responsabilidad del programador usar índices válidos!

No se pueden copiar arrays directamente (~~`array1 = array2`~~)

Hay que copiarlos elemento a elemento

Luis Hernández Yáñez/Pablo Moreno Ger

Arrays y bucles for

Procesamiento de arrays

- ✓ Recorridos
- ✓ Búsquedas
- ✓ Ordenación

etcétera...

Recorrido de arrays con bucles for

Arrays: tamaño fijo → Bucles de recorrido fijo (for)

```
tVentas ventas;
double media, total = 0;
...
for (int i = 0; i < Dias; i++) {
   total = total + ventas[i];
}
media = total / Dias;
```

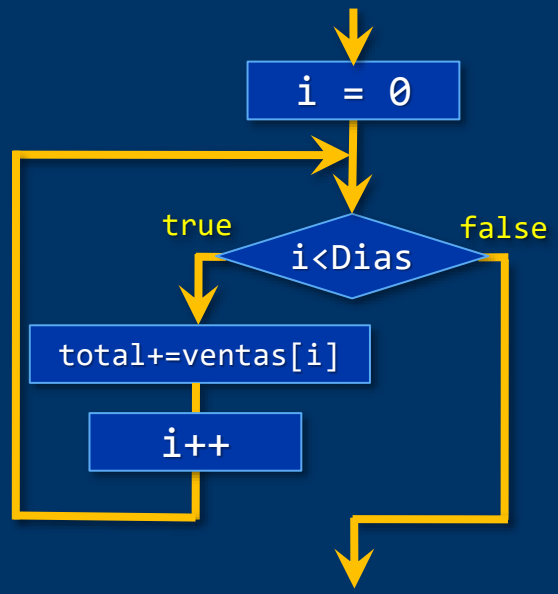
```
const int Dias = 7;
typedef double tVentas[Dias];
```

Luis Hernández Yáñez/Pablo Moreno Ger

Arrays y bucles for

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

```
tVentas ventas;
double media, total = 0;
...
for (int i = 0; i < Dias; i++) {
   total = total + ventas[i];
}
```



...

Memoria

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

Más sobre arrays

Luis Hernández Yáñez/Pablo Moreno Ger

Inicialización de arrays

Podemos inicializar los elementos de los arrays en la declaración

Asignamos una serie de valores al array:

```
const int DIM = 10;
typedef int tTabla[DIM];
tTabla i = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

Se asignan los valores por su orden:

```
i[0] i[1] i[2] i[3] i[4] ... i[9]
  ↑    ↑    ↑    ↑    ↑        ↑
 1º   2º   3º   4º   5º  ...  10º
```

Si hay menos valores que elementos, los restantes se ponen a `0`

```
tTabla i = { 0 }; // Pone todos los elementos a 0
```

Luis Hernández Yáñez/Pablo Moreno Ger

Enumerados como índices

```
const int Colores = 3,
typedef enum { rojo, verde, azul } tRGB;
typedef int tColor[Colores];
tColor color;
...
cout << "Cantidad de rojo (0-255): ";
cin >> color[rojo];
cout << "Cantidad de verde (0-255): ";
cin >> color[verde];
cout << "Cantidad de azul (0-255): ";
cin >> color[azul];
```

Recuerda que internamente se asignan enteros a partir de 0 a los distintos símbolos del enumerado

rojo ≡ 0 verde ≡ 1 azul ≡ 2

Luis Hernández Yáñez/Pablo Moreno Ger

Paso de arrays a subprogramas

Simulación de paso de parámetro por referencia

Sin poner & en la declaración del parámetro

Los subprogramas reciben la dirección en memoria del array

```
const int Max = 10;
typedef int tTabla[Max];
void inicializa(tTabla tabla); // Sin poner &
```

Las modificaciones del array quedan reflejadas en el argumento

```
inicializa(array);
```

Si `inicializa()` modifica algún elemento de `tabla`, automáticamente queda modificado ese elemento de `array`

¡Son el mismo array!

Luis Hernández Yáñez/Pablo Moreno Ger

Paso de arrays a subprogramas

```
const int Dim = 10;
typedef int tTabla[Dim];
void inicializa(tTabla tabla); // no se usa &

void inicializa(tTabla tabla) {
   for (int i = 0; i < Dim; i++)
      tabla[i] = i;
}
int main() {
   tTabla array;
   inicializa(array); // array queda modificado
   for (int i = 0; i < Dim; i++)
      cout << array[i] << " ";
   ...
```

```
0 1 2 3 4 5 6 7 8 9
```

Luis Hernández Yáñez/Pablo Moreno Ger

Paso de arrays a subprogramas

¿Cómo evitar que se modifique el array?

Usando el modificador `const` en la declaración del parámetro:

```
const tTabla tabla
```

Un array de constantes

```
void muestra(const tTabla tabla);
```

El argumento se tratará como un array de constantes

Si en el subprograma hay alguna instrucción que intente modificar un elemento del array: error de compilación

```
void muestra(const tTabla tabla) {
   for (int i = 0; i < Dim; i++) {
      cout << tabla[i] << " ";
      // OK. Se accede, pero no se modifica
   }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Implementación de listas

Luis Hernández Yáñez/Pablo Moreno Ger

Implementación de listas con arrays

Listas con un número fijo de elementos

Array con el nº de elementos como dimensión

```
const int NUM = 100;
typedef double tLista[NUM]; // Exactamente 100 double
tLista lista;
```

Recorrido de la lista:

```
for (int i = 0; i < NUM; i++) {
    ...
```

Búsqueda en la lista:

```
while ((i < NUM) && !encontrado) {
    ...
```

Luis Hernández Yáñez/Pablo Moreno Ger

Implementación de listas con arrays

Listas con un número variable de elementos

Array con un máximo de elementos + Contador de elementos

```
const int MAX = 100;
typedef double tLista[MAX]; // Hasta 100 elementos
tLista lista;
int contador = 0; // Se incrementa al insertar
```

Recorrido de la lista:

```
for (int i = 0; i < contador; i++) {
    ...
```

Búsqueda en la lista:

```
while ((i < contador) && !encontrado) {
    ...
```

¿Array y contador por separado? → Estructuras

Luis Hernández Yáñez/Pablo Moreno Ger

Cadenas de caracteres

Luis Hernández Yáñez/Pablo Moreno Ger

Cadenas de caracteres

Arrays de caracteres

Cadenas: secuencias de caracteres de longitud variable

```
"Hola" "Adiós" "Supercalifragilístico" "1234 56 7"
```

Variables de cadena: contienen secuencias de caracteres

Se guardan en arrays de caracteres: tamaño máximo (dimensión)

No todas las posiciones del array son relevantes:

- ✓ Longitud de la cadena: número de caracteres, desde el primero, que realmente constituyen la cadena:

Longitud actual: 4

Luis Hernández Yáñez/Pablo Moreno Ger

Cadenas de caracteres

Longitud de la cadena

Longitud: 5

Longitud: 21

Necesidad de saber dónde terminan los caracteres relevantes:

- ✓ Mantener la longitud de la cadena como dato asociado
- ✓ Colocar un carácter de terminación al final (*centinela*)

Luis Hernández Yáñez/Pablo Moreno Ger

Cadenas de caracteres

Cadenas de caracteres en C++

Dos alternativas para el manejo de cadenas:

- ✓ Cadenas al estilo de C (*terminadas en nulo*)
- ✓ Tipo `string`

Cadenas al estilo de C *Anexo del tema*

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

Tipo `string`

- ✓ Cadenas más sofisticadas
- ✓ Sin longitud máxima (gestión automática de la memoria)
- ✓ Multitud de funciones de utilidad (biblioteca `string`)

Luis Hernández Yáñez/Pablo Moreno Ger

Cadenas de caracteres de tipo `string`

Luis Hernández Yáñez/Pablo Moreno Ger

Cadenas de caracteres de tipo `string`

El tipo `string`

- ✓ El tipo asume la responsabilidad de la gestión de memoria
- ✓ Define operadores sobrecargados (+ para concatenar)
- ✓ Cadenas más eficientes y seguras de usar

Biblioteca `string`

Requiere establecer el espacio de nombres a `std`

- ✓ Se pueden inicializar en la declaración
- ✓ Se pueden copiar con el operador de asignación
- ✓ Se pueden concatenar con el operador +
- ✓ Multitud de funciones de utilidad

Luis Hernández Yáñez/Pablo Moreno Ger

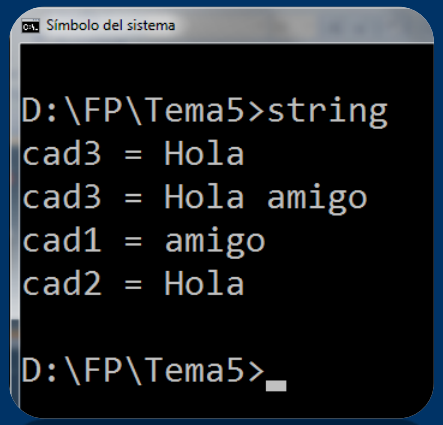
Cadenas de tipo string

string.cpp

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

int main() {
    string cad1("Hola");   // inicialización
    string cad2 = "amigo"; // inicialización
    string cad3;
    cad3 = cad1; // copia
    cout << "cad3 = " << cad3 << endl;
    cad3 = cad1 + " "; // concatenación
    cad3 += cad2;      // concatenación
    cout << "cad3 = " << cad3 << endl;
    cad1.swap(cad2);  // intercambio
    cout << "cad1 = " << cad1 << endl;
    cout << "cad2 = " << cad2 << endl;

    return 0;
}
```



Cadenas de tipo `string`

Longitud de la cadena:

`cadena.length()` o `cadena.size()`

Se pueden comparar con los operadores relacionales:

```
if (cad1 <= cad2) { ...
```

Acceso a los caracteres de una cadena:

- ✓ Como array de caracteres: *`cadena`*`[`*`i`*`]`
  Sin control de acceso a posiciones inexistentes del array
  Sólo debe usarse si se está seguro de que el índice es válido
- ✓ Función `at(`*`índice`*`)`: *`cadena`*`.at(`*`i`*`)`
  Error de ejecución si se accede a una posición inexistente

Luis Hernández Yáñez/Pablo Moreno Ger

E/S con cadenas de tipo `string`

- ✓ Se muestran en la pantalla con `cout <<`
- ✓ Lectura con `cin >>`: termina con espacio en blanco (inc. Intro)
  El espacio en blanco queda pendiente
- ✓ Descartar el resto de los caracteres del búfer:
  `cin.sync();`
- ✓ Lectura incluyendo espacios en blanco:
  `getline(cin, cadena)`
  Guarda en la *cadena* los caracteres leídos hasta el fin de línea
- ✓ Lectura de archivos de texto:
  Igual que de consola; `sync()` no tiene efecto
  `archivo >> cadena` `getline(archivo, cadena)`

Luis Hernández Yáñez/Pablo Moreno Ger

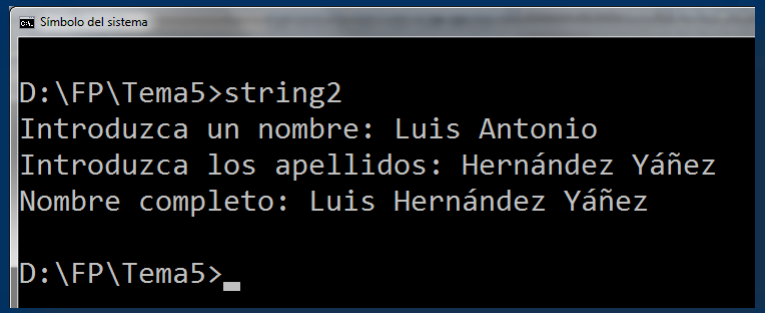
E/S con cadenas de tipo string

string2.cpp

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

int main() {
    string nombre, apellidos;
    cout << "Introduzca un nombre: ";
    cin >> nombre;
    cout << "Introduzca los apellidos: ";
    cin.sync();
    getline(cin, apellidos);
    cout << "Nombre completo: " << nombre << " "
         << apellidos << endl;

    return 0;
}
```



Luis Hernández Yáñez/Pablo Moreno Ger

Operaciones con cadenas de tipo string

- ✓ *cadena*.substr(*posición, longitud*)

  Subcadena de *longitud* caracteres desde *posición*

```
string cad = "abcdefg";
cout << cad.substr(2, 3); // Muestra cde
```

- ✓ *cadena*.find(*subcadena*)

  Posición de la primera ocurrencia de *subcadena* en *cadena*

```
string cad = "Olala";
cout << cad.find("la"); // Muestra 1
```

  (Recuerda que los arrays de caracteres comienzan con el índice 0)

- ✓ *cadena*.rfind(*subcadena*)

  Posición de la última ocurrencia de *subcadena* en *cadena*

```
string cad = "Olala";
cout << cad.rfind("la"); // Muestra 3
```

Luis Hernández Yáñez/Pablo Moreno Ger

Operaciones con cadenas de tipo string

- ✓ *cadena*.erase(*ini*, *num*)

  Elimina *num* caracteres a partir de la posición *ini*

```
string cad = "abcdefgh";
cad.erase(3, 4); // cad ahora contiene "abch"
```

- ✓ *cadena*.insert(*ini*, *cadena2*)

  Inserta *cadena2* a partir de la posición *ini*

```
string cad = "abcdefgh";
cad.insert(3, "123"); // cad ahora contiene "abc123defgh"
```

http://www.cplusplus.com/reference/string/string/

Luis Hernández Yáñez/Pablo Moreno Ger

Estructuras

Luis Hernández Yáñez/Pablo Moreno Ger

Estructuras

Colecciones heterogéneas (tuplas, registros)

Elementos de (posiblemente) distintos tipos: *campos*

Campos identificados por su nombre

Información relacionada que se puede manejar como una unidad

Acceso a cada elemento por su nombre de campo (operador `.`)

Luis Hernández Yáñez/Pablo Moreno Ger

Tipos de estructuras

```
typedef struct {
   ... // declaraciones de campos (como variables)
} tTipo; // nombre de tipo - ¡al final!
```

```
typedef struct {
   string nombre;
   string apellidos;
   int edad;
   string nif;
} tPersona;
```

Campos:

Tipos estándar o previamente declarado

Luis Hernández Yáñez/Pablo Moreno Ger

Variables de estructuras

```
tPersona persona;
```

Las variables de tipo `tPersona` contienen cuatro datos (campos):

`nombre` `apellidos` `edad` `nif`

Acceso a los campos con el operador punto (`.`):

```
persona.nombre // una cadena (string)
persona.apellidos // una cadena (string)
persona.edad   // un entero (int)
persona.nif // una cadena (string)
```

Podemos copiar dos estructuras directamente:

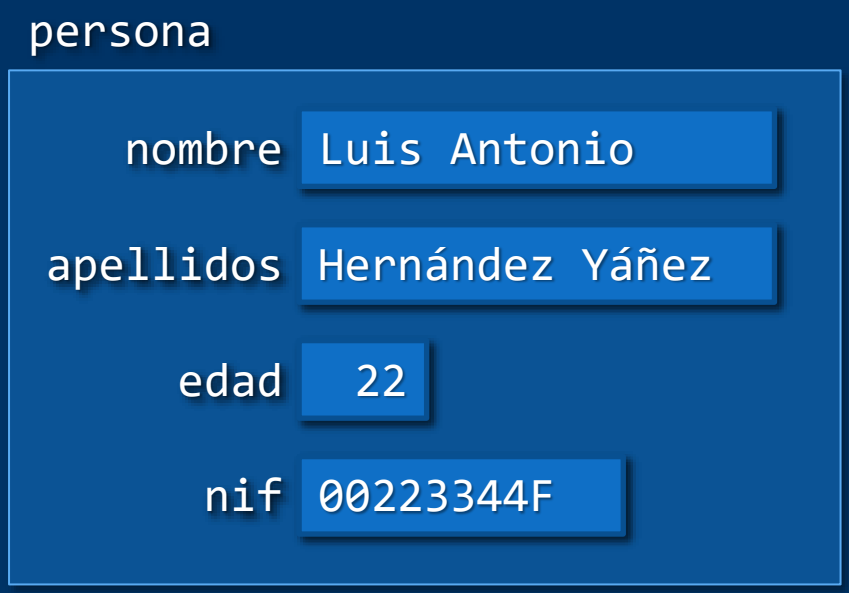
```
tPersona persona1, persona2;
...
persona2 = persona1;
```

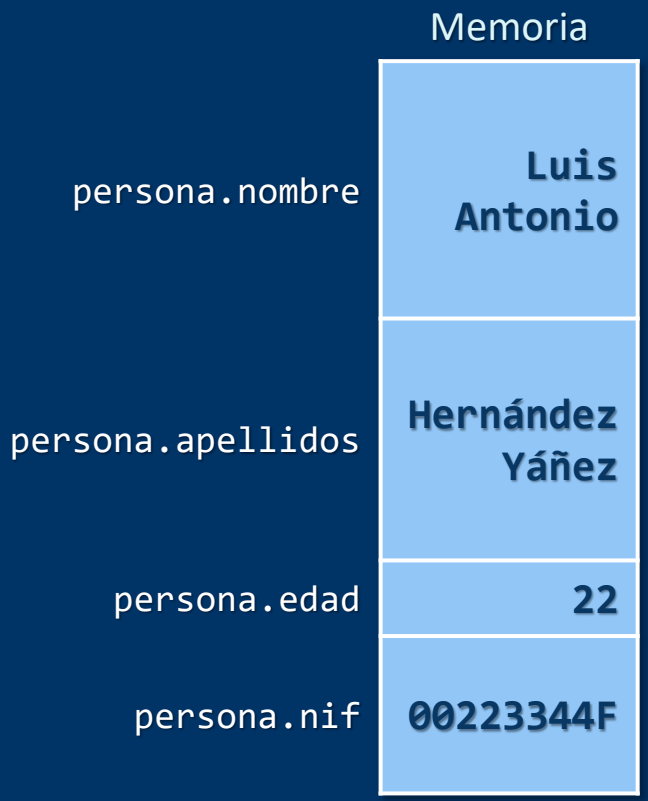
Se copian todos los campos a la vez

Luis Hernández Yáñez/Pablo Moreno Ger

Agrupación de datos heterogéneos

```
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
tPersona persona;
```





Elementos sin orden establecido

```
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
tPersona persona;
```

Los campos no siguen ningún orden establecido

Acceso directo por nombre de campo (operador .)

Con cada campo se puede hacer lo que permita su tipo

Las estructuras se pasan por valor (sin &)
o por referencia (con &) a los subprogramas

Luis Hernández Yáñez/Pablo Moreno Ger

Estructuras dentro de estructuras

```
typedef struct {
    string dni;
    char letra;
} tNif;
```

```
typedef struct {
    ...
    tNif nif;
} tPersona;
```

```
tPersona persona;
```

Acceso al NIF completo:

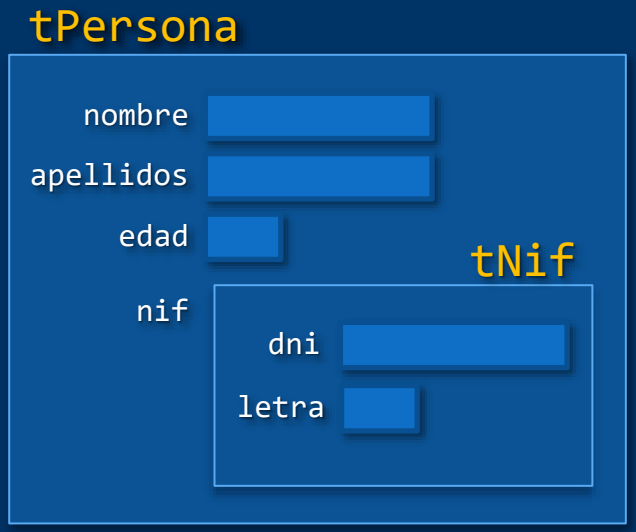
```
persona.nif // Otra estructura
```

Acceso a la letra del NIF:

```
persona.nif.letra
```

Acceso al DNI:

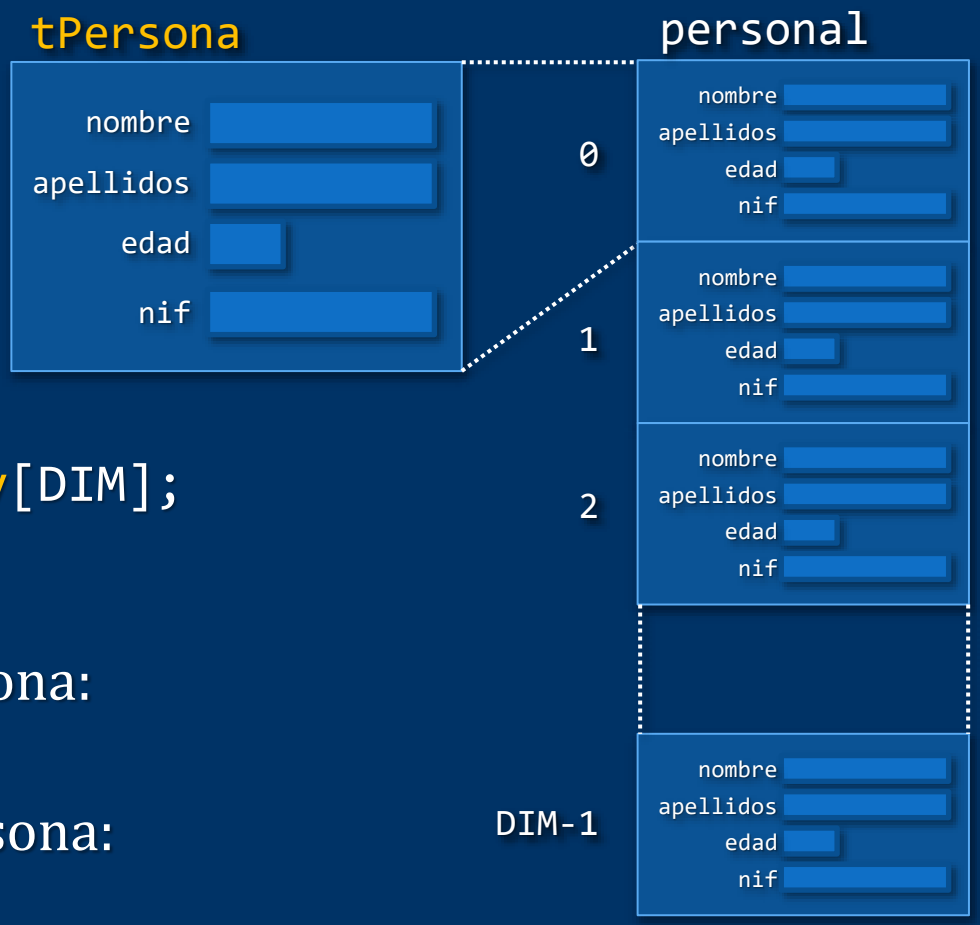
```
persona.nif.dni
```



Luis Hernández Yáñez/Pablo Moreno Ger

Arrays de estructuras

```
const int DIM = 100;
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
typedef tPersona tArray[DIM];
tArray personal;
```



Nombre de la tercera persona:
`personal[2].nombre`

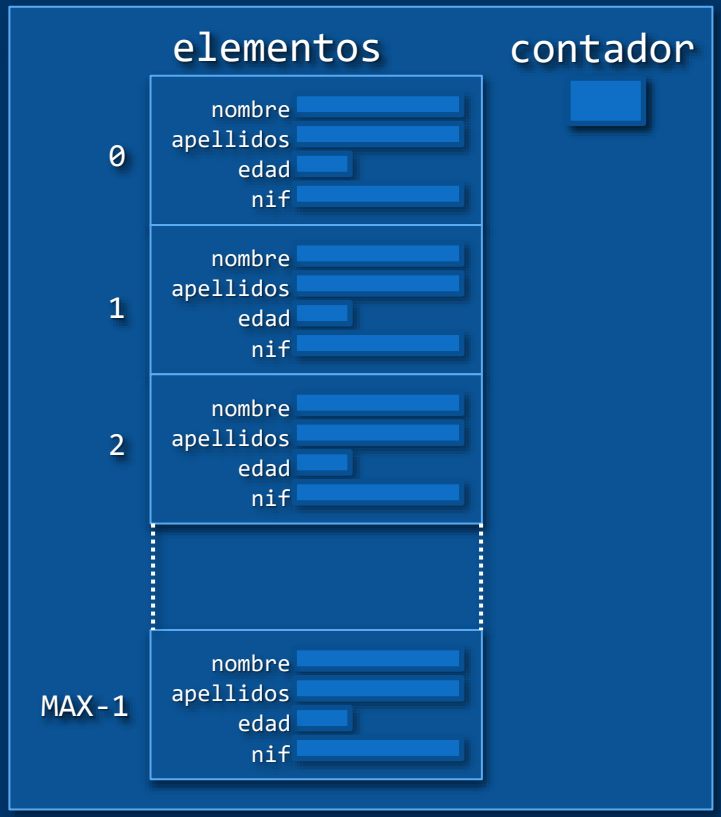
Edad de la duodécima persona:
`personal[11].edad`

NIF de la primera persona:
`personal[0].nif`

Luis Hernández Yáñez/Pablo Moreno Ger

Arrays dentro de estructuras

```
const int MAX = 100;
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
typedef tPersona tArray[MAX];
typedef struct {
    tArray elementos;
    int contador;
} tLista;
tLista lista;
```



Nombre de la tercera persona: `lista.elementos[2].nombre`

Edad de la duodécima persona: `lista.elementos[11].edad`

NIF de la primera persona: `lista.elementos[0].nif`

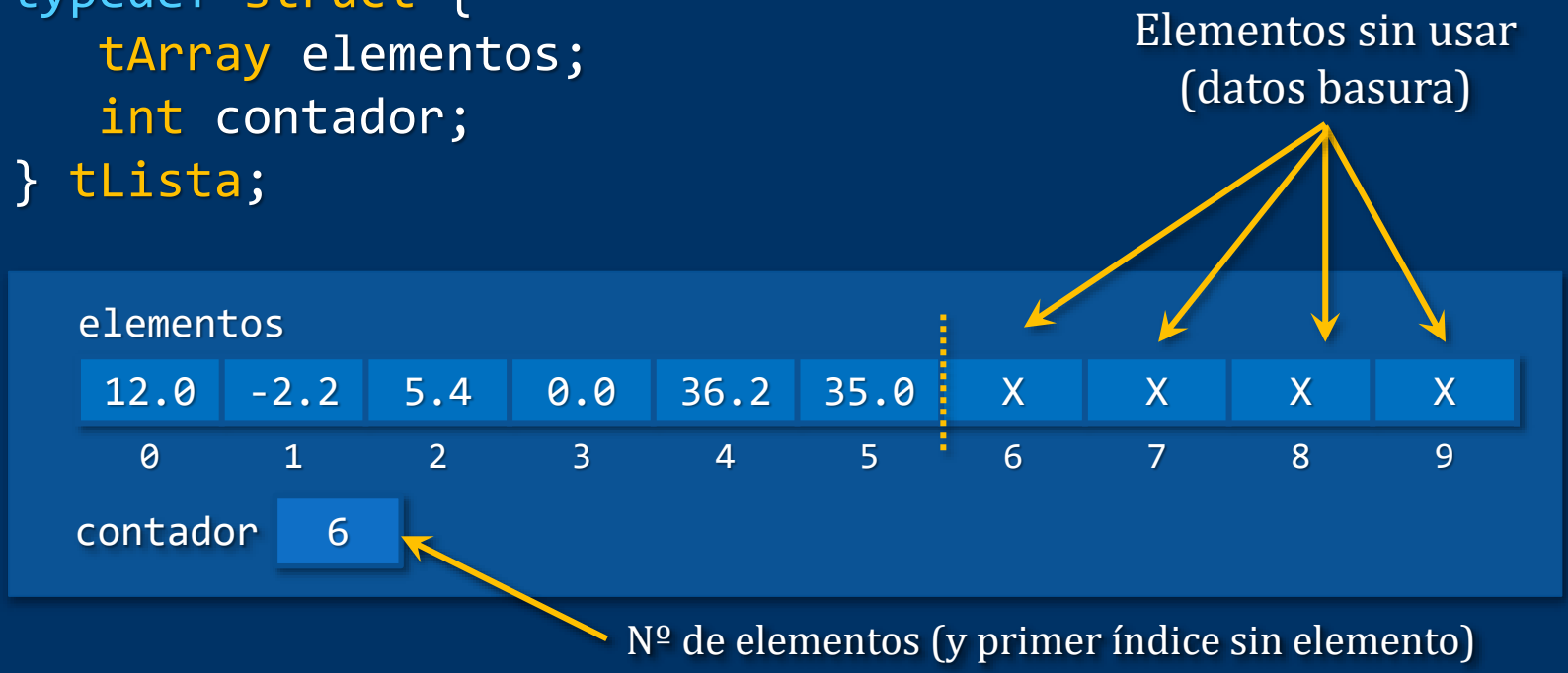
Listas de longitud variable

Luis Hernández Yáñez/Pablo Moreno Ger

Listas de longitud variable

Estructura que agrupe el array y el contador:

```
const int MAX = 10;
typedef double tArray[MAX];
typedef struct {
   tArray elementos;
   int contador;
} tLista;
```

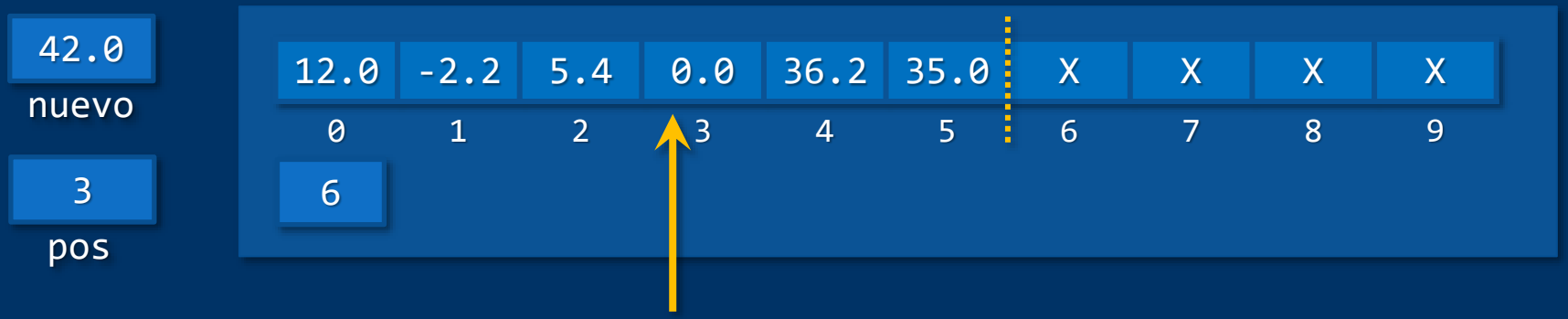


Operaciones principales: inserción y eliminación de elementos

Inserción de elementos

Insertar un nuevo elemento en una posición

Posiciones válidas: 0 a `contador`



Hay que asegurarse de que haya sitio (contador < máximo)

Operación en 3 pasos:

1.- Abrir hueco para el nuevo elemento (desde la posición)

2.- Colocar el elemento nuevo en la posición

3.- Incrementar el contador

Inserción de elementos

```
if (lista.contador < N) {
    // Abrir hueco
    for (int i = lista.contador; i > pos; i--) {
        lista.elementos[i] = lista.elementos[i - 1];
    }
    // Insertar e incrementar contador
    lista.elementos[pos] = nuevoElemento;
    lista.contador++;
}
```

Eliminación de elementos

Eliminar el elemento en una posición

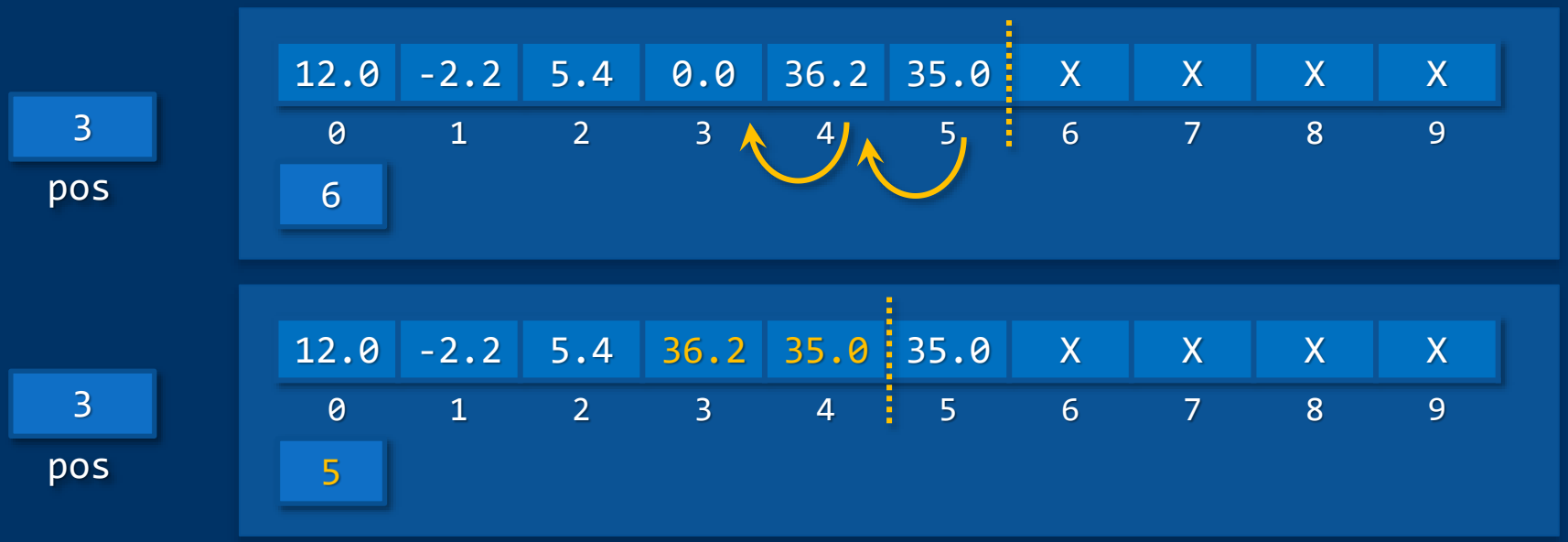
Posiciones válidas: 0 a `contador-1`

Desplazar a la izquierda desde el siguiente y decrementar el contador:

```
for (int i = pos; i < lista.contador - 1 ; i++) {
    lista.elementos[i] = lista.elementos[i + 1];
}
lista.contador--;
```


Eliminación de elementos

```
for (int i = pos; i < lista.contador - 1 ; i++) {
    lista.elementos[i] = lista.elementos[i + 1];
}
lista.contador--;
```



Un ejemplo completo

Luis Hernández Yáñez/Pablo Moreno Ger

Ejemplo de lista de longitud variable

Descripción

Programa que mantenga una lista de los estudiantes de una clase

De cada estudiante: nombre, apellidos, edad, NIF y nota

- ✓ Se desconoce el número total de estudiantes (máximo 100)
- ✓ La información de la lista se mantiene en un archivo `clase.txt`

  Se carga al empezar y se guarda al finalizar
- ✓ El programa debe ofrecer estas opciones:
  - Añadir un nuevo alumno
  - Eliminar un alumno existente
  - Calificar a los estudiantes
  - Listado de notas, identificando la mayor y la media

Luis Hernández Yáñez/Pablo Moreno Ger

Ejemplo de lista de longitud variable

bd.cpp

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

const int MAX = 100;
typedef struct {
   string nombre;
   string apellidos;
   int edad;
   string nif;
   double nota;
} tEstudiante;
typedef tEstudiante tArray[MAX];
typedef struct {
   tArray elementos;
   int contador;
} tLista;
```

Declaraciones de constantes y tipos globales

Tras las bibliotecas

Luis Hernández Yáñez/Pablo Moreno Ger

Ejemplo de lista de longitud variable

```
// Prototipos
int menu(); // Menú del programa - devuelve la opción elegida
void cargar(tLista &lista, bool &ok); // Carga del archivo
void guardar(const tLista &lista); // La guarda en el archivo
void leerEstudiante(tEstudiante &estudiante); // Lee los datos
void insertarEstudiante(tLista &lista, tEstudiante estudiante,
   bool &ok); // Inserta un nuevo estudiante en la lista
void eliminarEstudiante(tLista &lista, int pos, bool &ok);
// Elimina el estudiante en esa posición
string nombreCompleto(tEstudiante estudiante);
void calificar(tLista &lista); // Notas de los estudiantes
double mediaClase(const tLista &lista); // Nota media
int mayorNota(const tLista &lista);
// Índice del estudiante con mayor nota
void mostrarEstudiante(tEstudiante estudiante);
void listado(const tLista &lista, double media, int mayor);
// Listado de la clase
```

Los prototipos, después de los tipos globales

Luis Hernández Yáñez/Pablo Moreno Ger

El bucle do-while

Luis Hernández Yáñez/Pablo Moreno Ger

Otro bucle no determinado de C++

El bucle do..while

```
do cuerpo while (condición);
```

Condición al final del bucle


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

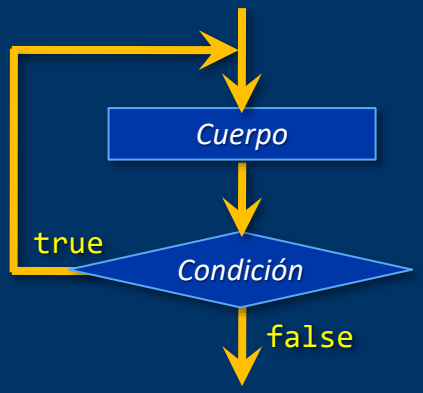
El *cuerpo* siempre se ejecuta al menos una vez

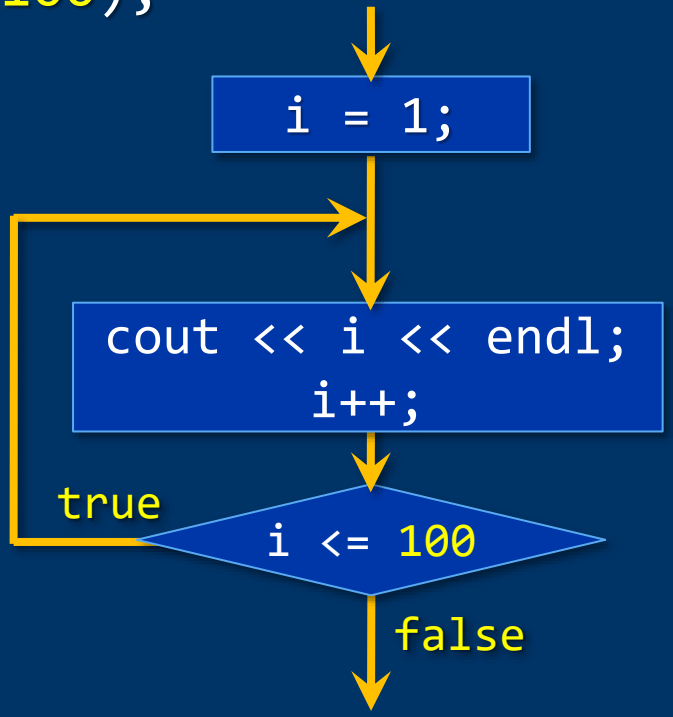
El *cuerpo* es un bloque de código

Luis Hernández Yáñez/Pablo Moreno Ger

Ejecución del bucle do-while

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





El cuerpo se ejecuta al menos una vez

Luis Hernández Yáñez/Pablo Moreno Ger

while *versus* do-while

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

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

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

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

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

Luis Hernández Yáñez/Pablo Moreno Ger

El menú de la aplicación con do-while

```
int menu() {
   int op;

   do {
      cout << "1 - Añadir un nuevo estudiante" << endl;
      cout << "2 - Eliminar un estudiante" << endl;
      cout << "3 - Calificar a los estudiantes" << endl;
      cout << "4 - Listado de estudiantes" << endl;
      cout << "0 - Salir" << endl;
      cout << "Opción: ";
      cin >> op;
   } while ((op < 0) || (op > 4));

   return op;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Ejemplo de lista de longitud variable

El archivo `clase.txt`

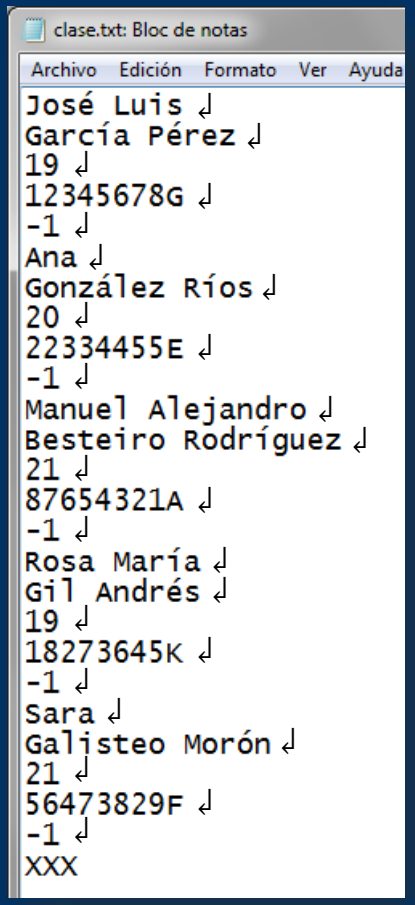
Un dato en cada línea

Por cada estudiante:

- ✓ Nombre (cadena)
- ✓ Apellidos (cadena)
- ✓ Edad (entero)
- ✓ NIF (cadena)
- ✓ Nota (real; -1 si no calificado)

Termina con XXX como nombre

El archivo se supone correcto



Ejemplo de lista de longitud variable

Lectura de la información de un estudiante

Nombre y apellidos:

Puede haber varias palabras → `getline()`

Edad → extractor (`>>`)

NIF: Una sola palabra → extractor (`>>`)

Nota → extractor (`>>`)

Queda pendiente de leer el Intro

Hay que saltar (leer) ese carácter con `get()`

Si no, en el siguiente nombre se leería una cadena vacía (Intro)

No leas directamente en la lista:

~~`getline(archivo, lista.elementos[lista.contador].nombre);`~~

Lee en una variable auxiliar de tipo `tEstudiante`

Carga del archivo clase.txt

```
void cargar(tLista &lista, bool &ok) {
   tEstudiante estudiante; // Variable auxiliar para leer
   ifstream archivo;
   char aux;
   lista.contador = 0; // Inicializamos la lista
   archivo.open("clase.txt");
   if (!archivo.is_open()) {
      ok = false;
   }
   else {
      ok = true;
      getline(archivo, estudiante.nombre); // Leemos el primer nombre
      while ((estudiante.nombre != "XXX") && (lista.contador < MAX)) {
         getline(archivo, estudiante.apellidos);
         archivo >> estudiante.edad;
         archivo >> estudiante.nif;
         archivo >> estudiante.nota;
         archivo.get(aux); // Saltamos el Intro
         lista.elementos[lista.contador] = estudiante; // Al final
         lista.contador++;
         getline(archivo, estudiante.nombre); // Siguiente nombre
      } // Si hay más de MAX estudiantes, ignoramos el resto
      archivo.close();
   }
}
```


Volcado en el archivo clase.txt

Simplemente, un dato en cada línea y en orden:

```
void guardar(const tLista &lista) {
   ofstream archivo;
   archivo.open("clase.txt");
   for (int i = 0; i < lista.contador; i++) {
      archivo << lista.elementos[i].nombre << endl;
      archivo << lista.elementos[i].apellidos << endl;
      archivo << lista.elementos[i].edad << endl;
      archivo << lista.elementos[i].nif << endl;
      archivo << lista.elementos[i].nota << endl;
   }
   archivo << "XXX" << endl; // Centinela final
   archivo.close();
}
```

`const tLista &lista` → Referencia constante
Paso por referencia pero como constante ≡ Paso por valor
Evita la copia del argumento en el parámetro (estructuras grandes)

Luis Hernández Yáñez/Pablo Moreno Ger

Lectura de los datos de un estudiante

```
void leerEstudiante(tEstudiante &estudiante) {
   cin.sync(); // Descartamos cualquier entrada pendiente
   cout << "Nombre: ";
   getline(cin, estudiante.nombre);
   cout << "Apellidos: ";
   getline(cin, estudiante.apellidos);
   cout << "Edad: ";
   cin >> estudiante.edad;
   cout << "NIF: ";
   cin >> estudiante.nif;
   estudiante.nota = -1; // Sin calificar de momento
   cin.sync(); // Descartamos cualquier entrada pendiente
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Inserción de un nuevo estudiante

```
void insertarEstudiante(tLista &lista, tEstudiante estudiante,
   bool &ok) {

   ok = true;
   if (lista.contador == MAX) {
      ok = false;
   }
   else {
      lista.elementos[lista.contador] = estudiante;
      // Insertamos al final
      lista.contador++;
   }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Eliminación de un estudiante

```
void eliminarEstudiante(tLista &lista, int pos, bool &ok) {
// Espera el índice del elemento en pos

   if ((pos < 0) || (pos > lista.contador - 1)) {
      ok = false; // Elemento inexistente
   }
   else {
      ok = true;
      for (int i = pos; i < lista.contador - 1; i++) {
         lista.elementos[i] = lista.elementos[i + 1];
      }
      lista.contador--;
   }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Calificación de los estudiantes

```
string nombreCompleto(tEstudiante estudiante) {
   return estudiante.nombre + " " + estudiante.apellidos;
}

void calificar(tLista &lista) {

   for (int i = 0; i < lista.contador; i++) {
      cout << "Nota del estudiante "
         << nombreCompleto(lista.elementos[i]) << ": ";
      cin >> lista.elementos[i].nota;
   }
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

Más subprogramas

```
double mediaClase(const tLista &lista) {
   double total = 0.0;
   for (int i = 0; i < lista.contador; i++) {
      total = total + lista.elementos[i].nota;
   }
   return total / lista.contador;
}

int mayorNota(const tLista &lista) {
   double max = 0;
   int pos = 0;
   for (int i = 0; i < lista.contador; i++) {
      if (lista.elementos[i].nota > max) {
         max = lista.elementos[i].nota;
         pos = i;
      }
   }
   return pos;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

El listado

```
void mostrarEstudiante(tEstudiante estudiante) {
   cout << setw(35) << left << nombreCompleto(estudiante);
   cout << estudiante.nif << " ";
   cout << setw(2) << estudiante.edad << " años ";
   cout << fixed << setprecision(1) << estudiante.nota;
}

void listado(const tLista &lista, double media, int mayor) {
   for (int i = 0; i < lista.contador; i++) {
      cout << setw(3) << i << ": ";
      mostrarEstudiante(lista.elementos[i]);
      if (i == mayor) {
         cout << " <<< Mayor nota!";
      }
      cout << endl;
   }
   cout << "Media de la clase: " << fixed << setprecision(1)
      << media << endl << endl;
}
```

Luis Hernández Yáñez/Pablo Moreno Ger

El programa principal

```
int main() {
   tLista lista;
   tEstudiante estudiante;
   bool exito;
   int op, pos;

   cargar(lista, exito);
   if (!exito) {
      cout << "No se ha podido cargar la lista!" << endl;
   }
   else {
      do { // El bucle do evita tener que leer antes la primera opción
         op = menu();
         switch (op) {
         case 1:
            {
               leerEstudiante(estudiante);
               insertarEstudiante(lista, estudiante, exito);
               if (!exito) {
                  cout << "Lista llena: imposible insertar" << endl;
               }
            }
            break;
```


El programa principal

```
            case 2:
               {
                  cout << "Posición: ";
                  cin >> pos;
                  eliminarEstudiante(lista, pos - 1, exito);
                  if (!exito) {
                     cout << "Elemento inexistente!" << endl;
                  }
               }
               break;
            case 3:
               {
                  calificar(lista);
               }
               break;
            case 4:
               {
                  listado(lista, mediaClase(lista), mayorNota(lista));
               }
            }
         } while (op != 0);
         guardar(lista);
      }
      return 0;
   }
```


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.

Luis Hernández Yáñez/Pablo Moreno Ger