

5

Tipos de datos estructurados

Grados en Ingeniería Informática, Ingeniería del Software e Ingeniería de Computadores

Alberto de la Encina Vara
(adaptadas del original de Luis Hernández Yáñez)



Facultad de Informática
Universidad Complutense



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 (posición): 0, 1, 2, 3, ...
- Una o varias dimensiones

Estructuras (tuplas, registros)

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



Fundamentos de la programación

Arrays



Arrays

Colecciones homogéneas

Mismo tipo de datos para todos los elementos:

- ✓ Notas de los estudiantes de una clase
 - ✓ Ventas de cada día de la semana
 - ✓ Temperaturas de cada día del mes
- etcétera...

En lugar de declarar N variables...

vLun	vMar	vMie	vJue	vVie	vSab	vDom
125.40	76.95	328.80	254.62	435.00	164.29	0.00

... declaramos una tabla de N variables:

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
Índices →	0	1	2	3	4	5	6

Arrays

Estructura indexada `tipo_base variable[num_elems];`

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

- ✓ Los índices son enteros positivos, de 0 a ($num_elems - 1$)
- ✓ El índice del primer elemento siempre es 0
- ✓ Los índices se incrementan de uno en uno

ventas	125.40	76.95	328.80	254.62	435.00	164.29	0.00
	0	1	2	3	4	5	6

Acceso directo `double ventas[7];`

A cada elemento se accede directamente a través de su índice:

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

`cout << ventas[4];`

`ventas[4] = 442.75;`



Cada elemento es un dato del tipo base
Se usa como cualquier otra variable

Tipos arrays

Declaración de tipos de arrays

```
typedef tipo_base tNombre[tamaño];
```



Ejemplos:

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

```
typedef short int tDiasMes[12];
```

```
typedef char tVocales[5];
```

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

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



Recuerda: los nombres de tipo los empezamos con una **t** minúscula, seguida de una o varias palabras, cada una con su inicial en mayúscula

Variables arrays

Declaración

tipo nombre;

Ejemplos:

```
tTemp tempMax;
```



```
tDiasMes diasMes;
```



```
tVocales vocales;
```



```
tVentas ventasFeb;
```



NO se inicializan los elementos automáticamente



Uso de arrays

Acceso a los elementos de un array

nombre[índice]

Cada elemento se accede a través del índice de su posición

```
tVocales vocales;
```

```
typedef char tVocales[5];
```

vocales	'a'	'e'	'i'	'o'	'u'
	0	1	2	3	4

5 elementos, con índices de 0 a 4:

```
vocales[0]  vocales[1]  vocales[2]  vocales[3]  vocales[4]
```

Como cualquier variable del tipo base:

```
cout << vocales[4];
```

```
vocales[3] = 'o';
```

```
if (vocales[i] == 'e')...
```

!!! ES VÁLIDO EL ÍNDICE???
¡Responsabilidad del programador!
!!! vocales[7] ??? !!!



Arrays y bucles for

Procesamiento de arrays

- ✓ Recorridos
- ✓ Búsquedas
- ✓ Ordenación etcétera...

Se verán con detenimiento en los siguientes temas

Recorridos de arrays con bucles for

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

```
tTemp tempMax;
```

```
double mediaMax, total = 0;
```

```
...
```

```
for (int i = 0; i < 7; i++)
```

```
    total = total + tempMax[i];
```

```
mediaMax = total / 7.0;
```

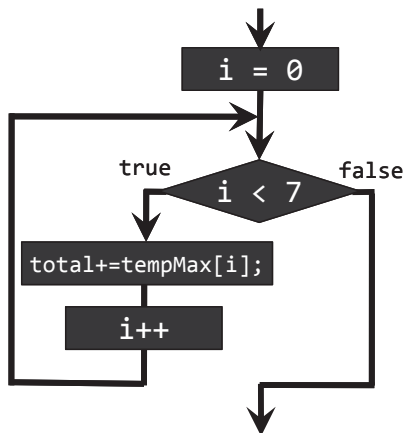
```
typedef double tTemp[7];
```



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

```
tTemp tempMax;  
double mediaMax, total = 0;  
...  
for (int i = 0; i < 7; i++)  
    total = total + tempMax[i];
```



	Memoria
→ tempMax[0]	12.40
→ tempMax[1]	10.96
→ tempMax[2]	8.43
→ tempMax[3]	11.65
→ tempMax[4]	13.70
tempMax[5]	13.41
tempMax[6]	14.07
mediaMax	?
total	84.62
i	7

Luis Hernández Yáñez



Fundamentos de la programación: Tipos de datos estructurados

Página 10



Inicialización de arrays

Podemos inicializar los elementos de los arrays en la declaración
Asignamos una lista de valores al array:

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Se asignan los valores por su orden:

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

```
for (int i = 0; i < 10; i++)  
    a[i] = i+1;
```

Copia de arrays

No se pueden copiar dos arrays (mismo tipo) :

No se puede hacer paso de parámetros por valor de arrays

No se puede hacer asignación de arrays

```
int array1[3] = {0, 0, 0};  
int array2[3];  
array2 = array1; // ¡¡¡ ERROR DE COMPILACION !!!
```

Han de copiarse los elementos uno a uno:

```
for (int i = 0; i < N; i++)  
    array2[i] = array1[i];
```

Paso de arrays a subprogramas

Sólo se pueden pasar por referencia (salida y entrada/salida)

Sin poner & en la declaración del parámetro

Los subprogramas reciben la dirección en memoria del array

```
const int Dim = 10;  
typedef int tTabla[Dim];  
void inicializa(tTabla tabla); // por referencia sin poner &
```

Las modificaciones del array quedan reflejadas en el argumento

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

¡Son el mismo array!

Paso de arrays a subprogramas

Un ejemplo

```
const int Dim = 10;
typedef int tTabla[Dim];
void inicializa(tTabla tabla); // no se usa &
// parámetro de salida, paso por referencia
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



Paso de arrays a subprogramas

¿Cómo evitar que se modifique el array?

Usando el modificador const en la declaración:

const tTabla tabla; declararíamos un array de constantes

Parámetro de entrada: por referencia constante

```
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 << array[i] << " ";
    // OK. Se accede, pero no se modifica
}
```


Enumerados como índices

```
typedef enum { rojo, verde, azul } tComps; // RGB
const int NumComps = 3;
typedef int tColor[NumComps];
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 \equiv 0 verde \equiv 1 azul \equiv 2

Capacidad de los arrays

Capacidad de un array

Definida en la declaración; no se puede cambiar en ejecución

Una decisión de diseño:

- ✓ En ocasiones será fácil (días de los 12 meses del año)
- ✓ Cuando pueda variar (listas) ha de estimarse un tamaño
Que no se quede corto ni desperdicie mucha memoria

Tema 9: memoria dinámica.

STL (*Standard Template Library*)



Ejemplos de arrays

Días de cada mes



Define el tamaño del array con una constante

```
const int Meses = 12;
typedef short int tDiasMes[Meses];

void inicializa(tDiasMes dias) { // por referencia, sin &
    dias[0] = 31; // Enero
    dias[1] = 28; // Febrero (¡no bisiesto!)
    dias[2] = 31; // Marzo
    dias[3] = 30; // Abril
    dias[4] = 31; // Mayo
    dias[5] = 30; // Junio
    dias[6] = 31; // Julio
    dias[7] = 31; // Agosto
    dias[8] = 30; // Septiembre
    dias[9] = 31; // Octubre
    dias[10] = 30; // Noviembre
    dias[11] = 31; // Diciembre
}
```



Los arrays se pasan siempre por referencia, sin usar & ¡dias vuelve modificado!

Ejemplos de arrays

Inicialización del array de días

```
int main() {
    tDiasMes diasMes; // Días de cada mes del año

    inicializa(diasMes); // diasMes queda modificado

    muestra(diasMes);
}
```

diasMes	31	28	31	30	31	30	31	31	30	31	30	31
	0	1	2	3	4	5	6	7	8	9	10	11



Los arrays se pasan siempre por referencia, sin usar & ¡dias vuelve modificado!

Ejemplos de arrays

```
// Muestra los días de cada mes del año
void muestra(const tDiasMes dias) { // por referencia constante
    for (int mes = 0; mes < Meses; mes++)
        cout << "Mes " << mes + 1 << ": " << diasMes[mes]
            << "días" << endl; ↑
}
```

Los usuarios usan de 1 a 12 para numerar los meses
La interfaz debe aproximarse a los usuarios,
aunque internamente se usen los índices de 0 a 11

```
// Calcula la media de días
double media(const tDiasMes dias) { // por referencia constante
    double total = 0;
    for (int mes = 0; mes < Meses; mes++)
        total = total + diasMes[mes];
    return total / Meses;
}
```

Ejemplos de arrays

Total de ventas de cada día del mes

```
const int MaxDias = 31;
typedef double tVentas[MaxDias];

¿Cuántos días tiene ese mes (febrero)?

const int Meses = 12;
typedef short int tDiasMes[Meses];

// Pide al usuario las ventas de un mes
void leeVentas(int diasMes, tVentas ventasMes) {
    for (int dia = 0; dia < diasMes; dia++) {
        cout << "Ventas del día " << dia + 1 << ": ";
        cin >> ventasMes[dia];
    }
}
```

Ejemplos de arrays

Total de ventas de cada día del mes

```
int main() {
    tDiasMes diasMes;
    tVentas ventasFeb; // Ventas de cada día de febrero

    inicializa(diasMes); // Asigna a cada mes su nº de días

    // Pedimos al usuario las ventas de ese mes...
    leeVentas(diasMes[1], ventasFeb);

    // ¡Ojo! Febrero tiene índice 1 (Enero el 0)

    ...
}
```

Ejemplos de arrays

Temperaturas máximas y mínimas de la semana

2 arrays paralelos que se manejan conjuntamente

Alternativas: array bidimensional o array de estructuras...

```
const int Dias = 7;
typedef double tTemp[Dias];
// Lee las temperaturas por consola
void leerTemp(tTemp tmpMin, tTemp tmpMax) {
    for (int dia = 0; dia < Dias; dia++) {
        cout << "Día " << dia + 1 << ": " << endl;
        cout << "Mínima: ";
        cin >> tmpMin[dia];
        cout << "Máxima: ";
        cin >> tmpMax[dia];
    }
}

int main() {
    tTemp tmpMin, tmpMax; // Mínimas y máximas - 2 arrays
```



Ejemplos de arrays

Array de valores de enumerado

```
string aCadena(tMoneda moneda); // Prototipo de función
const int MaxMonedas = 15;
typedef enum { centimo, dos_centimos, cinco_centimos,
             diez_centimos, veinte_centimos, medio_euro, euro }
             tMoneda;
typedef tMoneda tCalderilla[MaxMonedas];
tCalderilla bolsillo; // MaxMonedas monedas que llevo
bolsillo[0] = euro;
bolsillo[1] = cinco_centimos;
bolsillo[2] = medio_euro;
bolsillo[3] = euro;
bolsillo[4] = centimo;
...
for (int moneda = 0; moneda < MaxMonedas; moneda++)
    cout << aCadena(bolsillo[moneda]) << endl;
```



Fundamentos de la programación

Cadenas de caracteres



Cadenas de caracteres

Arrays de caracteres

Cadenas: secuencias de caracteres de longitud variable

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

Variables de cadena: contienen secuencias de caracteres

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

No todas las posiciones del array son relevantes:

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

H	o	l	a																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Longitud actual: 4



Cadenas de caracteres

Longitud de la cadena

A	d	i	ó	s																	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Longitud: 5

S	u	p	e	r	c	a	l	i	f	r	a	g	i	l	í	s	t	i	c	o	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Longitud: 21

Necesidad de saber dónde terminan los caracteres relevantes:

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

A	d	i	ó	s	\0																
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21



Cadenas de caracteres

Cadenas de caracteres en C++

Dos alternativas para el manejo de cadenas:

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

Cadenas al estilo de C

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

Tipo `string`

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



Cadenas de caracteres al estilo de C

Arrays de caracteres terminado en el carácter nulo

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

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

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

Indica que en esa posición termina la cadena (exclusive)

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

En el array caben *dimensión*-1 caracteres significativos

Longitud máxima de la variable `cadena`: 14

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

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



Cadenas de caracteres al estilo de C

Entrada/salida por consola

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

Extractor: la lectura termina en el primer separador

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

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

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

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

También lee espacios en blanco

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



Cadenas de caracteres al estilo de C

Funciones (biblioteca cstring)

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

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

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

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

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

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



Ejemplo de cadenas al estilo de C

```
#include <iostream>
using namespace std;
#include <cstring>
int main() {
    const int MAX = 20;
    typedef char tCad[MAX];
    tCad cadena = "Me gusta C++";
    cout << cadena << endl;
    cout << "Cadena: ";
    cin >> cadena; // Lee hasta el primer espacio en blanco
    cout << cadena << endl;
    cin.sync(); // Sincronizar la entrada
    cout << "Cadena: ";
    cin.getline(cadena, MAX);
    cout << cadena << endl;
    cout << "Longitud: " << strlen(cadena) << endl;
    strcpy(cadena, "Hola");
    ...
}
```

cadenas.cpp



Ejemplo de cadenas al estilo de C

```
tCad cadena2 = " amigo";
strcat(cadena, cadena2);
cout << cadena << endl;
if (strcmp(cadena, cadena2) == 0)
    cout << "Iguales";
else if (strcmp(cadena, cadena2) > 0)
    cout << cadena << " es mayor que " << cadena2;
else
    cout << cadena << " es menor que " << cadena2;
cout << endl;

return 0;
}
```

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

D:\FP\Tema5>
```



Cadenas de caracteres de tipo string

El tipo string

- ✓ El tipo asume la responsabilidad de la gestión de memoria
- ✓ Define operadores sobrecargados (p.e., + 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 (asignación, paso de parámetros por valor)
- ✓ Se pueden comparar con los operadores: ==, <, <=, >, >=
- ✓ Se pueden concatenar con el operador +
- ✓ Multitud de funciones de utilidad

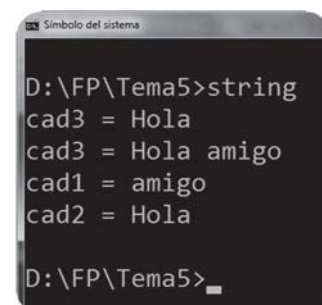
Cadenas de caracteres de tipo string

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

string.cpp

```
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;
}
```



```
Simbolo del sistema
D:\FP\Tema5>string
cad3 = Hola
cad3 = Hola amigo
cad1 = amigo
cad2 = Hola
D:\FP\Tema5>
```

```
D:/fb/16w92>
```




Cadenas de caracteres de tipo string

E/S con cadenas de tipo string

- ✓ Se muestran en la pantalla con `cout <<`
- ✓ Lectura con `cin >>`: termina con separador
- ✓ Descartar el resto de los caracteres del búfer:
`cin.sync(); cin.ignore(INT_MAX, '\n');`
- ✓ Lectura incluyendo espacios en blanco:
`getline(cin, cadena);`

Guarda en la *cadena* los caracteres leídos hasta el fin de línea

	<code>cin.getline(cad, max)</code>	Cadenas al estilo de C
	<code>getline(cin, cad)</code>	Cadenas de tipo string

Cadenas de caracteres de tipo string

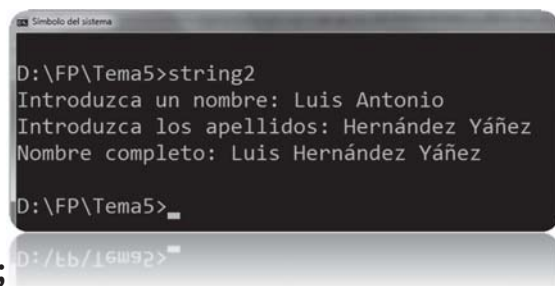
E/S con cadenas de tipo string

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;
}
```



Cadenas de caracteres de tipo string

Funciones para cadenas de tipo string Notación punto (.)

- ✓ Longitud de la cadena: `cadena.length()` / `cadena.size()`
- ✓ Acceso a los caracteres de una cadena

Recuerda que los índices comienzan en 0.

- ❖ Como array de caracteres: `cadena[i]`
Sin control de acceso a posiciones inexistentes del array
- ❖ Función `at(índice)`: `cadena.at(i)`
Error de ejecución si se accede a una posición inexistente

- ✓ Añadir eliminar caracteres al final de la cadena

- ❖ `push_back(char)`
- ❖ `pop_back();`

Modifican la longitud de la cadena

Cadenas de caracteres de tipo string

Funciones para cadenas de tipo string Notación punto (.)

- ✓ `substr(posición, Longitud)`
Subcadena de *longitud* caracteres desde *posición*

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

- ✓ `compare(cadena2)`: 0 si las cadenas son iguales, 1 si *cadena2* es menor que la cadena receptora y -1 si *cadena2* es mayor

```
string cad1 = "Hola", cad2 = "Adiós";  
cout << cad1.compare(cad2); // Muestra 1
```

Operadores relaciones: `==`, `<`, `<=`, `>`, y `>=`

- ✓ `find(subcadena)`: posición en la que empieza la primera ocurrencia de la *subcadena* en la cadena receptora

```
string cad = "Hola";  
cout << cad.find("la"); // Muestra 2
```

Cadenas de caracteres de tipo string

Funciones para cadenas de tipo string

- ✓ `rfind(subcadena)`: posición en la que empieza la última ocurrencia de la *subcadena* en la cadena receptora

```
string cad = "Olala";  
cout << cad.rfind("la"); // Muestra 3
```
- ✓ `find_first_of(cadena2)`: posición en la que aparece por primera vez cualquier carácter de *cadena2* en la cadena receptora

```
cout << cad.find_first_of("aeiou"); // Muestra 2
```
- ✓ `c_str()`: devuelve la cadena de caracteres al estilo C

```
cout << cad.c_str(); // Muestra "Olala"
```
- ✓ `stoi(cadena)`: si la cadena es un literal entero, lo devuelve como valor `int`

Cadenas de caracteres de tipo string

Más sobre cadenas de tipo string

- ✓ Modificación de cadenas:

```
cadena.erase(0, 7); // Elimina 7 caracteres desde el 1º  
cadena.replace(9, 5, cad2); // Reemplaza 5 caracteres a  
                             // partir del 9º por cad2  
cadena.insert(0, cad3); // Inserta en la posición 0 cad3  
cadena.append(3, '!'); // Añade al final 3 caracteres  
                       // de signo de exclamación (!)
```

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

- ✓ Parámetros de tipo `string` :
De entrada: por valor o por referencia constante (`const &`)
De salida o entrada/salida: por referencia (poniendo `&`)

Estructuras



Estructuras

Colecciones heterogéneas (tuplas, registros)

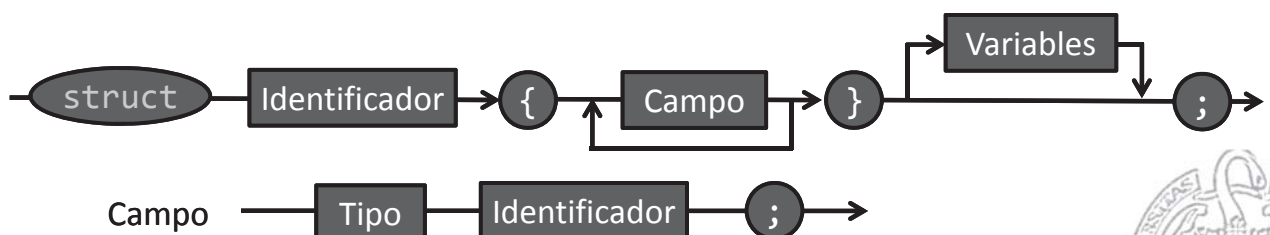
Elementos de (posiblemente) distintos tipos: *campos*

Campos identificados por su nombre

Información relacionada que se puede manejar como una unidad

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

```
struct nombre { // nombre de tipo
    tipo1 nombre_de_campo1;
    tipo2 nombre_de_campo2;
    ...
} lista_de_variables; // Puede no haber lista de variables
```



Tipos de estructuras

```
typedef struct {  
    ...  
} tTipo; // nombre de tipo al final  
  
typedef struct {  
    string nombre;  
    string apellidos;  
    int edad;  
    string nif;  
} tPersona;
```

Usamos el tipo para declarar variables: `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.edad` // un entero (int)



Agrupación de datos heterogéneos

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

persona

nombre	Luis Antonio
apellidos	Hernández Yáñez
edad	22
nif	00223344F

Memoria

persona.nombre

Luis
Antonio

persona.apellidos

Hernández
Yáñez

persona.edad

22

persona.nif

00223344F



Agrupación de datos heterogéneos

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



Las estructuras se pasan por valor
o por referencia a los subprogramas

Acceso directo por nombre de campo (operador .)

Con cada campo se puede hacer lo que permita su tipo

Estructuras dentro de estructuras:

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

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

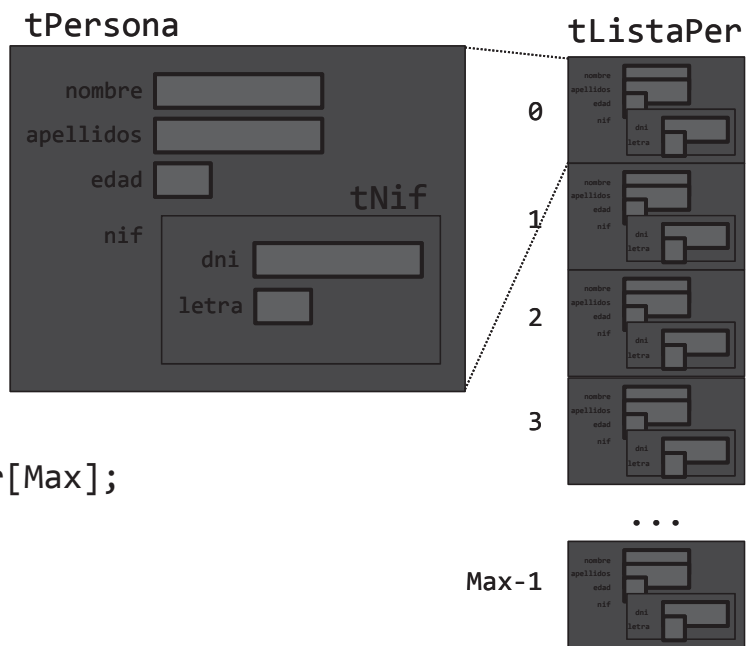
Fundamentos de la programación

Combinación de tipos estructurados



Combinación de tipos estructurados

```
const int Max = 100;
typedef struct {
    string dni;
    char letra;
} tNif;
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    tNif nif;
} tPersona;
typedef tPersona tListaPer[Max];
```



Combinación de tipos estructurados

```
const int Max = 100;
typedef struct {
    string dni;
    char letra;
} tNif;
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    tNif nif;
} tPersona;
typedef tPersona tListaPer[Max];
typedef double tListaNotas[Max];
typedef struct {
    tListaPer datos;
    tListaNotas notas;
    int cont;
} tClase;
```



Combinación de tipos estructurados

bd.cpp

```
#include <iostream>
#include <string>
using namespace std;
#include <iomanip>
const int Max = 100;
typedef struct {
    string dni;
    char letra;
} tNif;
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    tNif nif;
} tPersona;
typedef tPersona tListaPer[Max];
typedef double tListaNotas[Max];
typedef struct {
    tListaPer datos;
    tListaNotas notas;
    int cont;
} tClase;
```

Declaraciones de constantes y tipos
al principio del programa

Alcance global

(Si algún tipo o constante sólo se usa
en un subprograma, se declarará
dentro de ese subprograma)

Luis Hernández Yáñez



Fundamentos de la programación: Tipos de datos estructurados

(continúa...)

Página 50



Combinación de tipos estructurados

```
int menu(); // Menú del programa - devuelve la opción elegida
void inicializa(tClase &clase); // Lista vacía: contador a cero
bool dniOK(const string & dni); // ¿8 dígitos?
void leeNif(tNif & nif);
string toString(const tNif & nif); //NIF en forma de cadena
string nombreCompleto(const tPersona &persona); // Nombre + " "+ apell.
void leePersona(tPersona &persona);
double leeNota(const tPersona &persona); // Lee y devuelve la nota
bool insertaPersona(tClase &clase,const tPersona &persona,double nota);
// Inserta en la lista de la clase la persona y su nota
void escribePersona(const tPersona & persona);
// Muestra los datos de la persona en una línea (sin salto de línea)
double mediaClase(const tClase & clase); // Nota media de la clase
int mayorNota(const tClase & clase); // Índice de la mayor nota
void listado(const tClase & clase); // Lista de la clase
```

Los prototipos, después de los tipos globales

(continúa...)

Luis Hernández Yáñez



Fundamentos de la programación: Tipos de datos estructurados

Página 51



Combinación de tipos estructurados

```
int menu() {
    int op;
    do {
        cout << "1 - Nuevo estudiante" << endl;
        cout << "2 - Listado de notas" << endl;
        cout << "0 - Salir" << endl;
        cout << "Elige: ";
        cin >> op;
    } while ((op < 0) || (op > 2));
    return op;
}

void inicializa(tClase &clase) {
    clase.cont = 0; // Basta poner a 0 el contador
}

bool dniOK(const string &dni) {
    bool ok;
    ok = dni.size() == 8;
    if (ok) {
        int i = 0;
        while(i < 8 && ok)
            ok = ok && isDigit( dni.at(i) );
            i = i + 1 ;
    }
    return ok;
}
```

Búsqueda en cadena

(continúa...)

Combinación de tipos estructurados

```
void leeNif(tNif &nif) {
    do {
        cout << "D.N.I. (8 digitos): ";
        cin.sync();
        cin >> nif.dni;
    } while (!dniOK(nif.dni));
    do {
        cout << "Letra: ";
        cin.sync();
        cin >> nif.letra;
        nif.letra = toupper(nif.letra);
    } while ((nif.letra < 'A') || (nif.letra > 'Z'));
}

string toString(const tNif & nif) {
    return nif.dni + "-" + nif.letra; // Concatenación
} // Copia

string nombreCompleto(const tPersona & persona) {
    return persona.nombre + " " + persona.apellidos;
} // Copia
```

(continúa...)

Combinación de tipos estructurados

```
void leePersona(tPersona &persona) {
    cout << "Nombre (sólo uno): "; // Sólo nombres de una palabra
    cin.sync();
    cin >> persona.nombre;
    cout << "Apellidos: ";
    cin.sync();
    getline(cin, persona.apellidos);
    cout << "Edad: ";
    cin >> persona.edad;
    leeNif(persona.nif);
}

double leeNota(const tPersona &persona) {
    cout << "Nota de " << nombreCompleto(persona) << ": ";
    double nota;
    cin.sync();
    cin >> nota;
    return nota;
}
```

(continúa...)

Combinación de tipos estructurados

```
bool insertaPersona(tClase &clase,const tPersona &persona,double nota)
{
    bool ok;
    ok = (clase.cont != Max);
    if ( ok ) {
        clase.datos[clase.cont] = persona; // Copia
        clase.notas[clase.cont] = nota;
        clase.cont++;
    }
    return ok;
}

void escribePersona(const tPersona &persona) {
    cout << setw(25) << persona.nombre + " " + persona.apellidos
        << " (" << toString(persona.nif) << ", " << persona.edad
        << " años)";
}
```

(continúa...)

Combinación de tipos estructurados

```
double mediaClase(const tClase &clase) {
    double total = 0.0;
    for (int i = 0; i < clase.cont; i++)
        total = total + clase.notas[i];
    return total / clase.cont;
}
```

```
int mayorNota(const tClase &clase) {
    double max = 0;
    int pos = 0;
    for (int i = 0; i < clase.cont; i++)
        if (clase.notas[i] > max) {
            max = clase.notas[i];
            pos = i;
        }
    return pos;
}
```

(continúa...)

Combinación de tipos estructurados

```
void listado(const tClase &clase) {
    for (int i = 0; i < clase.cont; i++) {
        escribePersona(clase.datos[i]);
        cout << " - Nota: " << setw(4) << fixed
            << setprecision(1) << clase.notas[i] << endl;
    }
}
```

(continúa...)

Combinación de tipos estructurados

```
int main() {
    tClase clase;
    inicializa(clase);
    int op;
    do {
        op = menu();
        if (op == 1) {
            tPersona persona;
            leePersona(persona);
            double nota = leeNota(persona);
            if (!insertaPersona(clase, persona, nota))
                cout << "Lista llena: imposible insertar" << endl;
        }
        else if (op == 2) {
            int posMayorNota = mayorNota(clase);
            cout << "Mayor nota: " << clase.notas[posMayorNota] << endl;
        }
    } while (op != 0);

    return 0;
}
```

Combinación de tipos estructurados

```
D:\FP\Tema5>bd
1 - Nuevo estudiante
2 - Listado de notas
0 - Salir
Elige: 1
Nombre (solo uno): Luis
Apellidos: Hernández Yáñez
Edad: 20
D.N.I. (8 digitos): 00223344
Letra: g
Nota de Luis Hernández Yáñez (00223344-G, 20 años): 9.2
1 - Nuevo estudiante
2 - Listado de notas
0 - Salir
Elige: 1
Nombre (solo uno): Ana
Apellidos: Vegas García
Edad: 22
D.N.I. (8 digitos): 12345678
Letra: a
Nota de Ana Vegas García (12345678-A, 22 años): 9.6
1 - Nuevo estudiante
2 - Listado de notas
0 - Salir
Elige: 1
Nombre (solo uno): María
Apellidos: Vázquez Mella
Edad: 21
D.N.I. (8 digitos): 87654321
Letra: h
Nota de María Vázquez Mella (87654321-H, 21 años): 8.5
1 - Nuevo estudiante
2 - Listado de notas
0 - Salir
Elige: 2
Luis Hernández Yáñez (00223344-G, 20 años) - Nota: 9.2
Ana Vegas García (12345678-A, 22 años) - Nota: 9.6 <<< Mayor nota!
María Vázquez Mella (87654321-H, 21 años) - Nota: 8.5
```

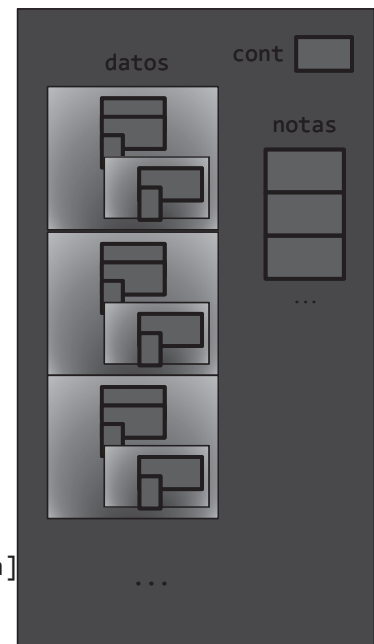


Combinación de tipos estructurados

```
int main() {
    tClase clase;
    inicializa(clase);
    int op;
    do {
        op = menu();
        if (op == 1) {
            tPersona persona;
            leePersona(persona);
            double nota = leeNota(persona);
            if (!insertaPersona(clase, persona, nota))
                cout << "Lista llena: imposible insertar"
                    << endl;
        }
        else if (op == 2) {
            int posMayorNota = mayorNota(clase);
            cout << "Mayor nota: " << clase.notas[posMayorNota]
                << endl;
        }
    } while (op != 0);

    return 0;
}
```

clase



Luis Hernández Yáñez

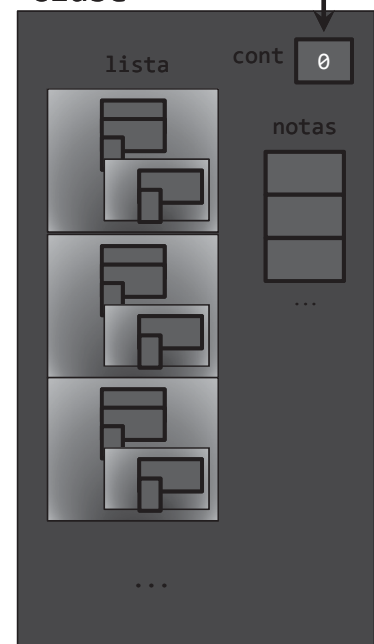


Combinación de tipos estructurados

```
int main() {
    tClase clase;
    inicializa(clase);
    int op;
    do {
        op = menu();
        if (op == 1) {
            tPersona persona;
            leePersona(persona);
            double nota = leeNota(persona);
            if (!insertaPersona(clase, persona, nota))
                cout << "Lista llena: imposible insertar"
                    << endl;
        }
        else if (op == 2) {
            int posMayorNota = mayorNota(clase);
            listado(clase, posMayorNota);
        }
    } while (op != 0);

    return 0;
}
```

clase



Luis Hernández Yáñez



Combinación de tipos estructurados



Combinación de tipos estructurados



Combinación de tipos estructurados

```
int main() {
    tClase clase;
    inicializa(clase);
    int op;
    do {
        op = menu();
        if (op == 1) {
            tPersona persona;
            leePersona(persona);
            double nota = leeNota(persona);
            if (!insertaPersona(clase, persona, nota))
                cout << "Lista llena: imposible insertar"
                    << endl;
        }
        else if (op == 2) {
            int posMayorNota = mayorNota(clase);
            listado(clase, posMayorNota);
        }
    } while (op != 0);

    return 0;
}
```

persona

nombre	Luis
apellidos	Hernández
edad	22
nif	dni 00112233
	letra F

clase

lista	cont 0
	notas
	...

nota 9.7



Combinación de tipos estructurados

```
int main() {
    tClase clase;
    inicializa(clase);
    int op;
    do {
        op = menu();
        if (op == 1) {
            tPersona persona;
            leePersona(persona);
            double nota = leeNota(persona);
            if (!insertaPersona(clase, persona, nota))
                cout << "Lista llena: imposible insertar"
                    << endl;
        }
        else if (op == 2) {
            int posMayorNota = mayorNota(clase);
            listado(clase, posMayorNota);
        }
    } while (op != 0);

    return 0;
}
```

persona

nombre	Luis
apellidos	Hernández
edad	22
nif	dni 00112233
	letra F

clase

lista	cont 0
	notas
	...

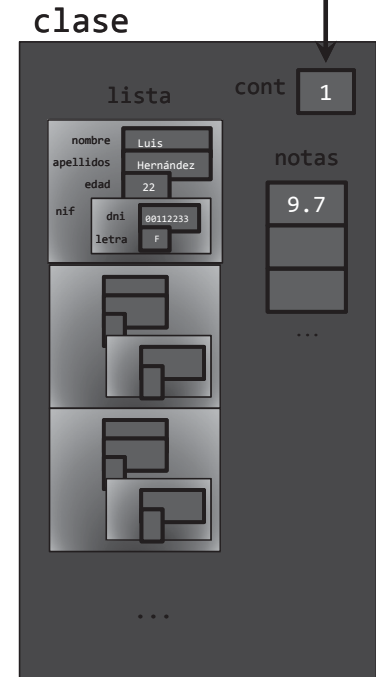
nota 9.7



Combinación de tipos estructurados

```
int main() {
    tClase clase;
    inicializa(clase);
    int op;
    do {
        op = menu();
        if (op == 1) {
            tPersona persona;
            leePersona(persona);
            double nota = leeNota(persona);
            if (!insertaPersona(clase, persona, nota))
                cout << "Lista llena: imposible insertar"
                    << endl;
        }
        else if (op == 2) {
            int posMayorNota = mayorNota(clase);
            listado(clase, posMayorNota);
        }
    } while (op != 0);

    return 0;
}
```



nota 9.7



Referencias bibliográficas

- ✓ *Programming. Principles and Practice Using C++*
B. Stroustrup. Pearson Education, 2009
- ✓ *Programación en C++ para ingenieros*
F. Xhafa et al. Thomson, 2006
- ✓ *C++: An Introduction to Computing (2ª edición)*
J. Adams, S. Leestma, L. Nyhoff. Prentice Hall, 1998
- ✓ *El lenguaje de programación C++ (Edición especial)*
B. Stroustrup. Addison-Wesley, 2002








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.

