Programación I

Arrays, cadenas de caracteres y estructuras

Iván Cantador
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Contenidos

- **Arrays**
- Cadenas de caracteres
- Estructuras
- Enumeraciones

Arrays (I)

- Un **array** (*tabla*, *vector*) es una colección ordenada de datos de un mismo tipo
  - Declaración de variable

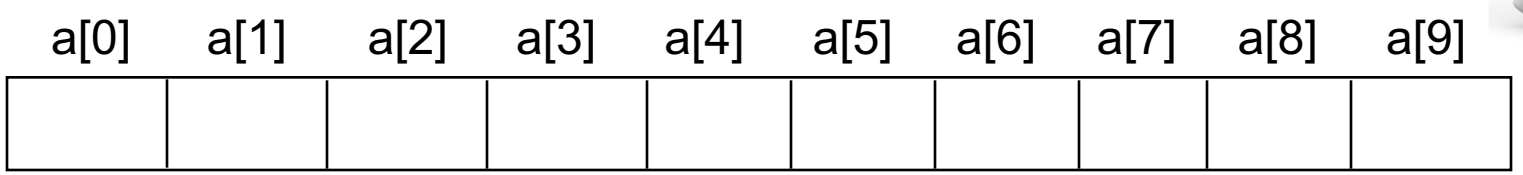
```
<tipo_dato> <nombre>[tamanio];

int a[3];                       // Un array de 3 int
float b[4], c[5];
```

  - Inicialización (en la declaración):

```
int a[3] = {1, 2, 3};
float b[] = {1, 2, 3, 4}; /* Se puede omitir el tamanio */
float b[5] = {1, 2, 3};   /* Se inicializan los 3 primeros
                             valores */
```

Arrays (II)

- El primer elemento de un array **a** de *N* elementos es **a[0]**, el segundo es **a[1]**, ..., el último es **a[*N*-1]**
  - Ejemplo: **array de 10 elementos**



En C:

```
int a[10];
a[0] = 2;          // Guardamos un 2 en la primera posicion
a[9] = 4;          // Guardamos un 4 en la ultima posicion
a[10] = 8;         // ERROR! Se ha excedido la longitud del array
scanf("%d", &a[5]);    // Leemos por teclado la posición 6
```

Arrays (III)

- La asignación entre arrays **NO** está definida en C
  - Solución: recorrer los elementos de los arrays mediante un bucle

```
int a[] = {1, 2, 3};      // Declaracion CON inicializacion
int b[3];                 // Declaracion SIN inicializacion
int i;

// b = a; es incorrecto
// Para hacerlo se usara un bucle
for ( i=0; i<3; i++ ) {
   b[i] = a[i];
}
```

Arrays (IV)

- **Arrays multidimensionales** (*matrices*)

```
int m1[2][3];                              // Declaracion SIN inicializacion
int m2[][3] = {{1, 2, 3}, {4, 5, 6}};      // Declaracion CON inicializacion
int i, j;

// Inicializacion de todas las celdas a 0 mediante un bucle
for (i=0; i<2; i++) {
    for (j=0; j<3; j++) {
        m1[i][j] = 0;
    }
}
// Asignacion de valores a celdas (0,0) y (1,2)
m1[0][0] = 1;
m1[1][2] = 2;

// Impresion por pantala de todas las celdas mediante un bucle
for (i=0; i<2; i++) {
    for (j=0; j<3; j++) {
        printf("%d, ", m1[i][j]);
    }
    printf("\n");
}
```

Contenidos

- Arrays
- **Cadenas de caracteres**
- Estructuras
- Enumeraciones

Cadenas de caracteres (I)

- **Cadenas de caracteres (*strings*)**
  - En C una cadena de caracteres es un **array de char**, donde **el último carácter es el *carácter de fin de cadena* '\0'**

```
// Las siguientes 3 declaraciones son equivalentes
char s1[] = "Hola";
char s2[] = {'H', 'o', 'l', 'a', '\0'};
char *s3 = "Hola";

// Modificamos el segundo caracter de la cadena s1
printf("El valor de s1 es: %s.\n", s1); // Hola
s1[1] = 'A';
printf("El valor de s1 es: %s.\n", s1); // HAla
```

Cadenas de caracteres (II)

- **Lectura por teclado de cadenas**
  - Lectura de una cadena por teclado con **scanf**, usando **%s** (la s viene de *string*) → NO hay que poner &

```
char cadena[256];

printf("Introduce tu nombre: ");
scanf("%s", cadena);          // Atencion! No se pone &cadena
printf("Hola %s!\n", cadena);
```

Cadenas de caracteres (III)

- **Algunas funciones básicas sobre cadenas (string.h)**
  - `int strlen(char *s)`
    - devuelve la longitud de s (SIN contar el '\0' del final de s)
  - `char *strcpy(char *s1, char *s2)`
    - copia s2 en s1
    - devuelve s1 (modificada)
  - `char *strcat(char *s1, char *s2)`
    - concatena s2 al final de s1
    - devuelve s1 (modificada)
  - `int strcmp(char *s1, char *s2)`
    - devuelve un número < 0 si s1 < s2
    - devuelve un número > 0 si s1 > s2
    - devuelve 0 si s1 = s2

    → La comparación de cadenas es carácter a carácter, con sus códigos ASCII

Cadenas de caracteres (IV)

- **Algunas funciones básicas sobre cadenas (string.h)**

```
char cadena1[256] = "hola";
char cadena2[256] = "mundo!";
char cadena3[256];

printf("La longitud de cadena1 es %d\n", strlen(cadena1)); // 4

strcpy(cadena3, cadena1);  // cadena3 vale "hola"

strcat(cadena3, cadena2);  // cadena3 vale "holamundo!"

printf("%d\n", strcmp(cadena1, cadena1)); // Imprime 0
printf("%d\n", strcmp(cadena1, cadena2)); // Imprime numero < 0
printf("%d\n", strcmp(cadena2, cadena1)); // Imprime numero > 0
```

Contenidos

- Arrays
- Cadenas de caracteres
- **Estructuras**
- Enumeraciones

Estructuras (I)

- **Una estructura (*struct*)** es una colección de datos de (tal vez) diferentes tipos → a los datos se les suelen llamar atributos o campos

  - Declaración **struct**

```
struct {
   <atributo1>;
   <atributo2>;
   ...
   <atributoN>;
} <nombreEstructura>;
```

Ejemplo:

```
#define MAX_TAM 256
struct {
   char titulo[MAX_TAM];
   char autor[MAX_TAM];
   int anio;
} Libro;
```

  - Declaración de variable

```
struct <nombreEstructura> <variable>;
```

Ejemplo:

```
void main() {
   ...
   struct Libro l;
   ...
}
```

Estructuras (II)

- **Una estructura (*struct*)** es una colección de datos de (tal vez) diferentes tipos → a los datos se les suelen llamar atributos o campos

  - Declaración con **typedef**

```
typedef struct {
   <atributo1>;
   <atributo2>;
   ...
   <atributoN>;
} <nombreEstructura>;
```

Ejemplo:

```
#define MAX_TAM 256
typedef struct {
   char titulo[MAX_TAM];
   char autor[MAX_TAM];
   int anio;
} Libro;
```

  - Declaración de variable

```
<nombreEstructura> <variable>;
```

Ejemplo:

```
void main() {
   ...
   Libro l;
   ...
}
```

Estructuras (III)

- El tamaño (en Bytes) de una estructura es la suma de los tamaños de sus atributos

```
#define MAX_TAM 256
`
typedef struct {
   char titulo[MAX_TAM];
   char autor[MAX_TAM];
   int anio;
} Libro;

// sizeof(Libro) = 2 * 256 * sizeof(char) + sizeof(int)
```

Estructuras (IV)

- **Acceso a los campos de una estructura**: se realiza de la forma `nombreVariable.nombreCampo`

```
#include <stdio.h>
#include <string.h>

#define MAX_TAM 256
typedef struct {
   char titulo[MAX_TAM];
   char autor[MAX_TAM];
   int anio;
} Libro;

void main() {
   // Definicion de una variable de tipo Libro
   Libro l;

   // Acceso a los campos de la variable
   strcpy(l.titulo, "El Señor de los Anillos");
   strcpy(l.autor, "J.R.R. Tolkien");
   l.anio = 1958;
   printf("%s escrito por %s en %d\n", l.titulo, l.autor, l.anio);
}
```

Estructuras (V)

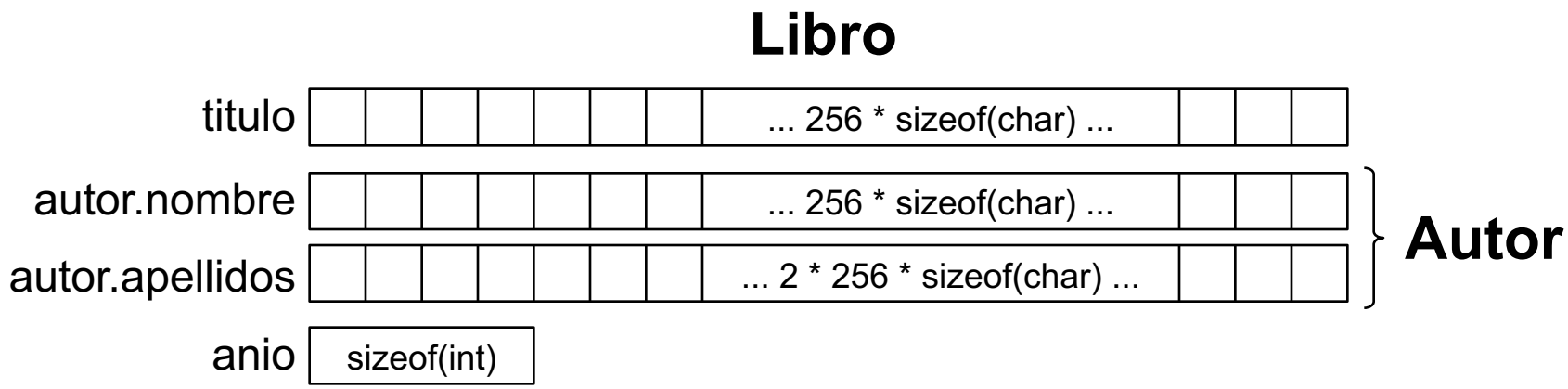
- **Estructuras anidadas** – declaración

```
#include <stdio.h>
#include <string.h>

#define MAX_TAM 256
typedef struct {
    char nombre[MAX_TAM];
    char apellidos[2*MAX_TAM];
} Autor;

typedef struct {
    char titulo[MAX_TAM];
    Autor autor;      // La estructura Autor debe estar declarada antes/arriba
    int anio;
} Libro;

void main() {
    Libro l;

    strcpy(l.titulo, "El Señor de los Anillos");
    strcpy(l.autor.nombre, "J.R.R.");
    strcpy(l.autor.apellidos, "Tolkien");
    l.anio = 1958;
    printf("%s escrito por %s %s en %d\n", l.titulo, l.autor.nombre,
                                           l.autor.apellidos, l.anio);
}
```

Estructuras (VI)

- **Estructuras anidadas** – espacio en memoria

```
#define MAX_TAM 256
typedef struct {
   char nombre[MAX_TAM];
   char apellidos[2*MAX_TAM];
} Autor;

typedef struct {
   char titulo[MAX_TAM];
   Autor autor;
   int anio;
} Libro;
```



Estructuras (VII)

- **Arrays de estructuras**

```
#include <stdio.h>
#include <string.h>
#define MAX_TAM 256
typedef struct {
   char titulo[MAX_TAM];
   char autor[MAX_TAM];
   int anio;
} Libro;
void main() {
   Libro libros[2];
   strcpy(libros[0].titulo, "Don Quijote de la Mancha");
   strcpy(libros[0].autor, "Miguel de Cervantes Saavedra");
   libros[0].anio = 1605;
   strcpy(libros[1].titulo, "Hamlet");
   strcpy(libros[1].autor, "William Shakespeare");
   libros[1].anio = 1601;
   printf("Libro: %s de %s.\n", libros[0].titulo, libros[0].autor);
   printf("Libro: %s de %s.\n", libros[1].titulo, libros[1].autor);
}
```

Estructuras (VIII)

- **Ejercicio.** Modificar la estructura Libro (y si es necesario definir nuevas estructuras) para:
  - almacenar información (p.e. nombre y descripción) sobre una lista de géneros literarios fija
  - asociar a cada libro uno o más de los géneros literarios almacenados

```
#define MAX_CAD 256
typedef struct {
   char titulo[MAX_CAD];
   char autor[MAX_CAD];
   int anio;
} Libro;
```

Estructuras (IX)


```c
#include <stdio.h>
#include <string.h>

#define MAX_CAD 256
#define MAX_TXT 2048
#define MAX_GEN 8

typedef struct {
    char nombre[MAX_CAD];
    char descripcion[MAX_TXT];
} Genero;

typedef struct {
    char titulo[MAX_CAD];
    char autor[MAX_CAD];
    int anio;
    Genero generos[MAX_GEN];
    int numGeneros;
} Libro;
```

```c
void main() {
    Libro l;
    int i;

    // Guardamos los datos del libro
    strcpy(l.titulo, "El Señor de los Anillos");
    strcpy(l.autor, "J.R.R. Tolkien");
    l.anio = 1958;

    strcpy(l.generos[0].nombre, "novela");
    l.numGeneros = 1;
    strcpy(l.generos[1].nombre, "fantasía");
    l.numGeneros++;
    strcpy(l.generos[2].nombre, "ficción");
    l.numGeneros++;

    // Sacamos por pantalla los datos del libro
    printf("Título: %s\n", l.titulo);
    printf("Autor: %s\n", l.autor);
    printf("Año: %d\n", l.anio);

    printf("Géneros:\n");
    for ( i=0; i<l.numGeneros; i++ ) {
        printf("\t%s\n", l.generos[i].nombre);
    }
}
```

Contenidos

- Arrays
- Cadenas de caracteres
- Estructuras
- **Enumeraciones**

Enumeraciones (I)

- Un **enum** (enumeración) es un tipo de dato que permite agrupar valores constantes identificados con nombres dados
  - Declaración

```
enum {<nombre_1>, <nombre_2>, ...} <nombre_enum>;

enum {<nombre_1>=<valor_1>, <nombre_2>=<valor_2>, ...} <nombre_enum>;
```

  → Si no se indican los valores de las constantes, estos por defecto son 0, 1, 2, …

  - Se suele utilizar **typedef** para definir tipos de datos enum

```
typedef enum {<nombre_1>, <nombre_2>, ...} <nombre_enum>;

typedef enum {<nombre_1>=<valor_1>, <nombre_2>=<valor_2>, ...} <nombre_enum>;
```

Enumeraciones (II)

- Un **enum** (enumeración) es un tipo de dato que permite agrupar valores constantes identificados con nombres dados
  - Ejemplo (sin typedef)

```
#include <stdio.h>

enum {verde, amarillo, rojo} Colores;

void main() {
   enum Colores semaforo;

   semaforo = verde;        // equivale a semaforo = 0
   semaforo = amarillo;     // equivale a semaforo = 1
   semaforo = rojo;         // equivale a semaforo = 2

   printf("Color del semaforo: %d\n", semaforo);
}
```

Enumeraciones (III)

- Un **enum** (enumeración) es un tipo de dato que permite agrupar valores constantes identificados con nombres dados
  - Ejemplo (con typedef)

```
#include <stdio.h>

typedef enum {verde, amarillo, rojo} Colores;

void main() {
   Colores semaforo;

   semaforo = verde;        // equivale a semaforo = 0
   semaforo = amarillo;     // equivale a semaforo = 1
   semaforo = rojo;         // equivale a semaforo = 2

   printf("Color del semaforo: %d\n", semaforo);
}
```

Enumeraciones (IV)

- Un **enum** (enumeración) es un tipo de dato que permite agrupar valores constantes identificados con nombres dados
  - Ejemplo

```
#include <stdio.h>

typedef enum {verde=5, amarillo=3, rojo=8} Colores;

void main() {
   Colores semaforo;

   semaforo = verde;  // equivale a semaforo = 5

   printf("Color del semaforo: %d\n", semaforo);
}
```

Enumeraciones (V)

- Un **enum** (enumeración) es un tipo de dato que permite agrupar valores constantes identificados con nombres dados
  - Ejemplos "clásicos"

```
typedef enum {ERROR=-1, OK=0} status;

typedef enum {FALSE=0, TRUE=1} boolean;


void main() {
   status estadoEjecucion;
   boolean finEjecucion;

   estadoEjecucion = OK;
   finEjecucion = FALSE;
}
```