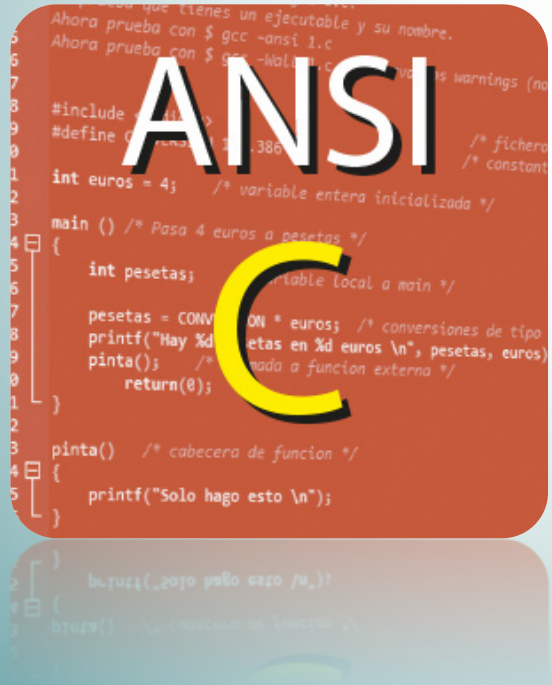
Programación Estructurada en ANSI C

Sesión 4



Rafael Menéndez de Llano Rozas

DEPARTAMENTO DE INFORMÁTICA Y ELECTRÓNICA

Índice

4. Datos estructurados

- Existen en C dos tipos de datos estructurados:
  - Colección del mismo tipo de datos: **array**.
  - Colección de distinto tipo de datos: **struct**.
- Para la definición del array se utiliza el operador primario **[ ]** con el número total de elementos:

```
int array_de_enteros[100];
```

- Los elementos del array se recorren con un índice que siempre empieza en cero (ojo) y es entero:

```
array_de_enteros[25] = 321;
```

- En C99 se pueden hacer arrays con asignación dinámica que se crean en tiempo de ejecución. Muy cómodo.
- También se pueden inicializar con llaves (no hace falta static):

```
static int dias[] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

  - Se puede omitir el límite. Si se pone puede coincidir, o no. Si es menor se ponen a cero, si es mayor da error.

4. Comparación de arrays

- Las siguientes declaraciones de arrays en Java:

```
final int MAX=8;
float[ ] v1;                    //índice va de 0 a MAX-1
v1=new float[MAX];      // tamaño puesto en la creación
short[ ] [ ] c=new short[MAX][MAX];
```

- Tienen el siguiente equivalente en C:

```
#define MAX 8            // constante en C
float v1[MAX];           //índice va de 0 a MAX-1array
// creado al declararlo
short int c [MAX][MAX];
```

- En C los arrays de números NO se inicializan a cero.

4. Datos estructurados: arrays

- El nombre de un array en C es sinónimo de su **dirección** de comienzo, por lo tanto es como un puntero constante.
- Las direcciones de los arrays no se pueden cambiar, son constantes.
- Las operaciones con punteros son (algunas ya vistas):
  - Asignación.
  - Comparación.
  - Dirección e Indirección.
  - Incremento y Decremento.
  - Adicción y Diferencia.
- Ejercicio 17: bájate el programas 23.c y guárdalo en tu directorio.
  1. Ábrelo con un editor, examínalo y compréndelo.
  2. Observa la **equivalencia** entre punteros y arrays.
  3. Mira las operaciones con punteros.

4. Datos estructurados: arrays

```
#include <stdio.h>
#define TAM 3
int main ()
{
   int fechas[TAM]={1,4,7}, *pun, indice, suma;
   double  facturas[TAM]={1.0,5.0,8.0}, *punf;

   pun = fechas;          /* asignacion de punteros */
   // fechas=pun;         /* array = array + 3;  <=>   5 = 5 + 3; */
   punf = facturas;
   printf("tamaños %lu %lu %lu \n", sizeof(int), sizeof(double), sizeof(fechas));
   for (indice = 0; indice < TAM; indice++)
      printf("punteros + %d: %10p %10p \n", indice, pun + indice, punf + indice);
   for (indice = 0; indice < TAM; indice++)
      printf("indice  %d: %d  %d %lf  %lf\n", indice, fechas[indice],
                              *(pun+indice), facturas[indice], *(punf+indice));
   for (indice = 0, suma = 0; indice < TAM; indice++)
      suma += *(pun + indice);
   printf("Suma: %d \n", suma);
   printf("Diferencia de dos punteros enteros distancia 2: %ld \n",
                                                  &facturas[2]-&facturas[0]);
   return(0);
}
```

4. Datos estructurados: arrays dobles

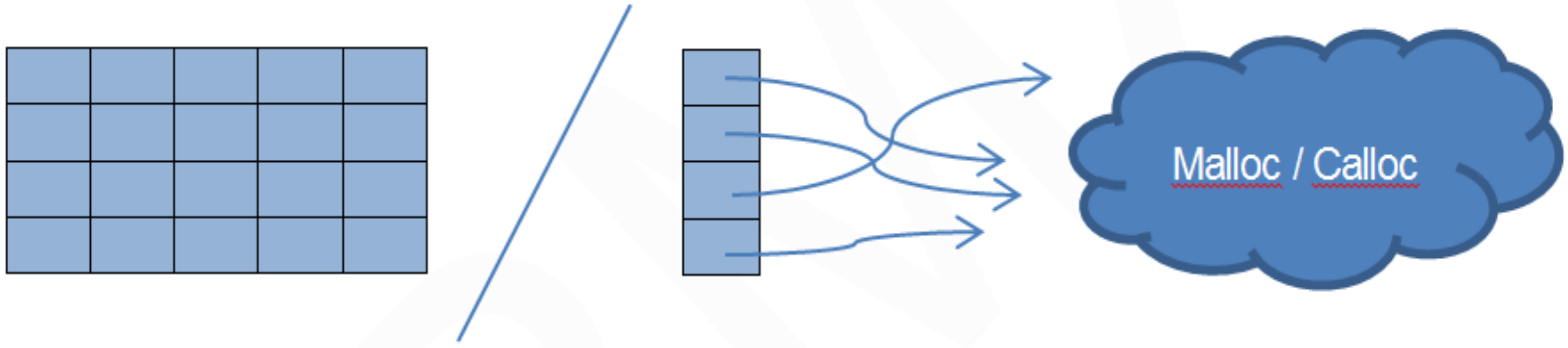
- También se pueden hacer de varias dimensiones:

```
int matriz[4][5];   //almacenados por filas
```

- Se pueden inicializar:

```
int matriz[2][2] = { {1,2}, {2,4} };
int matriz[2][2] = { 1, 2, 2, 4 };
```

- Si se omite una dimensión (*slice*) nos estamos refiriendo a toda la fila.
- En C no se puede usar matriz[2,2] ya que es usar el operador “,”.
- También son equivalentes a varias formas de punteros:
  - Una array de punteros.
  - Un puntero doble.
  - …



Ejercicio 18: Arrays dobles

- Bájate los programas 24.c, 25.c y 26.c guárdalos en tu directorio.
  1. Ábrelos con un editor.
  2. Observa como se accede a una matriz doble.
  3. Observa como se accede a un puntero doble.
  4. Y comprende las equivalencias entre punteros y que asignaciones están bien y mal.

4. Datos estructurados: string

- El string en C es un array (puntero) de caracteres terminados en uno especial (marca de fin de string) que es el carácter nulo '\0', para el que también se reserva espacio.
- El string nulo es NULL (en stdio.h).
- La inicialización se hace con comillas (string constante):
  - char tira[20] = "esto es un mensaje";
  - char otra[ ] = "esto es otro mensaje";
  - char *pun_tira = “Llevamos 9 horas";
- Los elementos son caracteres:
  - tira[12] = 's';
- Para entrada/salida se utiliza el conversor `%s` o las funciones puts() y gets() –esta última está obsoleta y es peligrosa– :

```
char tira[81];            /* declaracion de array */
pun_tira = gets(tira);
puts(tira);
```

4. Datos estructurados: string

- Hay cosas que no se pueden hacer:

```
char tira[], *pun;
…
tira = "esto no se puede hacer";      // es igual que hacer 23=15
pun = "esto si";                      // var = 15;
```

- Existe una librería para trabajar con string a la que se accede a través de **string.h**. Las funciones más usuales son:
  - **strcpy**(): toma como argumentos dos tiras de caracteres y copia la segunda en la primera, es decir sus contenidos, no sus direcciones. Hay que comprobar que la primera tiene espacio suficiente.
  - **strlen**(): toma una tira como argumento y devuelve un entero con su longitud sin contar el carácter nulo.
  - **strcat**(): toma como argumentos dos tiras de caracteres y devuelve en la primera la unión de ambas. Para ello debe asegurarse que en la primera tira caben las dos unidas.
  - **strcmp**(): toma como argumentos dos tiras de caracteres y compara sus contenidos, no sus direcciones. Devuelve un entero que indica si las tiras son iguales o no. En el primer caso devolverá un 0, en el segundo un entero positivo o negativo dependiendo de si la primera tira es mayor o menor que la segunda. El valor absoluto será la diferencia de código ASCII.

Ejercicio 19: Strings

- Bájate los programas 27.c , 28.c y 29.c guárdalos en tu directorio.
  1. Ábrelos con un editor.
  2. Observa como se trabaja con strings y las inicializaciones.
  3. Cambia la función gets por fgets.
  4. Usa la librería de manejo de strings .
  5. Date cuenta que el algoritmo es importante.

4. Datos estructurados: arrays y funciones

- Se pueden pasar arrays a funciones como argumentos. Por ejemplo un array de enteros (se puede poner la dimensión):
  - Prototipo: `int media(int array[ ], int);`
  - Llamada: `entero = media(array, n);`
  - Definición: `int media(int array[ ], int n) { … }`
- Al ser direcciones siempre se pasan por referencia salvo uso de `const`.
- No pueden devolverse array pero si punteros (o por argumento).
- Pasar como argumento un array de dos dimensiones es como pasar un puntero a un array. Al menos hay que dar una dimensión (se pueden dar las dos):
  - Dos: `función(int matriz [2][3]) …`
  - Una: `función(int matriz [ ][3]) …`
  - Puntero: `función(int (*matriz)[3]) …`

4. Datos estructurados: struct

- Es una colección de datos de distinto tipo. Parecido a la clase Java sin operaciones (sólo con datos).
- Primero hay que definir una plantilla (*template*), por ejemplo:

```
struct estructura                    /* hace de tipo */
{
        char nom_miembro1;     /* los campos de la struct */
        int  nom_miembro2;     /* cualquier tipo */
             ...               /* mas tipos … */
        char array[81];
} ;
```

Puede ir la variable directa

- Y después la declaración de una variable:

```
struct estructura tuya;                /* normal */
struct estructura {...} tuya;          /* todo en uno */
struct estructura mia = {'a', 23, …, "pepe"};  /* inicial */
struct estructura mia[100];     /* array de estructura */
struct estructura *deaquel;     /* puntero a estructura */
```

4. Datos estructurados: struct

- Puede ser muy práctico usar typedef:

```
typedef struct estructura Estructura;
typedef float complejo[2];
```

- Las operaciones permitidas con las structs son:
  - Tener acceso a sus miembros.
  - Obtener su dirección.
  - Asignarlas (copiarlas). Esto no estaba permitido en C clásico K&R.
- Para acceder a los campos se utiliza el operador "." o "->":

```
mia[32].nom_miembro1 = '2';
tuya.miembro2 = 13;
deaquel->array[5] = '3';    /* -> para punteros para evitar
   (*deaquel).array[5]
```

- Las dos últimas implican que se pueden pasar como argumentos a funciones (por valor) y se pueden devolver:
  - Prototipo: `struct comp suma(struct pri, struct seg);`
  - Llamada: `complejo=suma(A,B);`
  - Definición:

```
struct comp suma (struct comp uno,
                  struct comp dos)   { … }
```

Ejercicio 20: arrays, funciones y structs

- Bájate los programas 30.c y 31.c guárdalos en tu directorio.
  1. Ábrelos con un editor.
  2. Observa como se pasan arrays a funciones. Y como son equivalentes a punteros.
  3. Como por defecto se pasan por referencia pero también se puede hacer análogo a por valor.
  4. Analiza como se hacen las plantillas de los structs.
  5. Como se pueden pasar y devolver a/de funciones.
  6. Cambia el programa 31 y usa typedef.
  7. Descubre como se almacenan los campos de una struct.
  8. Prueba a poner entre dos campos otro de tipo char.