Programación I

Teoría III

http://proguno.unsl.edu.ar

proguno@unsl.edu.ar

DATOS ESTRUCTURADOS

Estructuras de Datos

- Hasta ahora hemos trabajado con …
  - Datos simples
    - enteros
    - reales
    - Caracteres
    - punteros
- Sin embargo, existe la necesidad de manipular datos compuestos
  - Relacionados lógicamente
  - Unidad conceptual

Estructuras de Datos

- Datos Estructurados
  - Arreglos
  - Registros
  - Pilas
  - Filas
  - Listas
  - Arboles …
- Ejemplos de uso:
  - Datos de una persona
  - Notas de examen de los alumnos de un curso.

Estructuras de Datos

- Al trabajar con datos estructurados, surgen nuevos interrogantes:
  - ¿*Cuántos* elementos se pueden almacenar? (*Capacidad*)
  - ¿Cómo se *incorpora, elimina o cambia y se recupera* un elemento de la estructura? (*Operaciones*)
  - ¿En qué *orden* se encuentran elementos, uno con respecto a los otros? (*Orden)*
  - ¿Cómo se identifican o *seleccionan* los elementos de la estructura? (*Selector*).
  - ¿Qué *tipo de datos* pueden guardarse en la estructura? (*Tipo Base*).

Veamos cada uno …

Capacidad de las Estructuras de Datos

- Dinámica
  - Crece y decrece con las inserciones y supresiones.
- Estática
  - Fija, no crece ni decrece.

Operaciones y Predicados de Control sobre las Estructuras de Datos

- COLOCAR / INSERTAR un elemento nuevo en la estructura / ASIGNAR en el caso de estructuras estáticas.
- SACAR / SUPRIMIR de la estructura un elemento ya existente en ella / ASIGNAR en el caso de estructuras estáticas.
- INSPECCIONAR / LEER / RECUPERAR un elemento de la estructura para conocer su valor.
- ¿Está LLENA?
- ¿Está VACÍA?
- Otras operaciones propias de cada tipo de estructura de datos.

Orden de los Elementos

- Orden cronológico
  - Ejemplos:
    - Orden de inserción
    - Orden de supresión
    - Orden de inspección
- Orden no cronológico
  - Ejemplos:
    - Orden alfabético, numérico, lexicográfico …

Selector de Elementos

- Selecciona, elige, identifica de forma unívoca cada uno de los elementos de la estructura.

- Explícito
  - El selector debe ser referenciado explícitamente en el código del programa al ser utilizado.
- Implícito
  - No es necesario definir un identificador para el selector. El elemento seleccionado será algún *elemento distinguido* de la estructura.

Tipo Base de una Estructura

- Es el tipo de los elementos de la estructura
- Puede ser:
  - Homogéneo / Heterogéneo
  - Simple / Compuesto o estructurado

ARREGLOS

Arreglos

- Estructura de datos homogénea donde todos sus elementos se encuentran contiguos en la memoria y se acceden a través de un índice.

- Selector: explícito (índice o suscripto).
  - Arreglos unidimensionales → un índice
  - Arreglos multidimensionales → múltiples índices

- Orden cronológico: No existe. El acceso es directo y aleatorio a cualquier elemento cuya dirección se puede calcular por una expresión. Podría existir otro.

Arreglos

- En general, la capacidad es estática. En algunos lenguajes hay arreglos dinámicos o extensibles.

- Operaciones:
  - Asignación
  - Inspección

Arreglos en **C**

- Son estáticos, no hay arreglos dinámicos.
- Tienen las mismas reglas de alcance (ámbito) que cualquier variable simple.
- Índices: expresiones que evalúen a un entero.
  - Comienzan siempre desde la posición 0.

-1	9	300	-54	0	23	4	11	-34	-7
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

```
int a[10];
int x = 2;
a[2+x] = a[2+x] + x;
```

Declaración de Arreglos en C

- Ejemplos de declaraciones válidas

```
int a[10];
int b[50], c[8];
int a[4] = {20, 30, 40, 10};
char b[10] = {‘a’, ‘B’, ‘?’};
float c[] = {3.1, 4.5, 2.6};
```

```
#include <stdio.h>
#define SIZE     10

int main(){
  int a[SIZE], i;

  for (i = 0 ; i < SIZE; i++)
      a[i] = 0;
  printf("Indice\tElemento\n");
  for (i = 0 ; i < SIZE; i++)
      printf("%d\t%d\n", i, a[i]);
  return 0;
}
```

Arreglos Multidimensionales en C

```
int a[2][3];
```

	Columna 0	*Columna* 1	*Columna* 2
Fila 0	`a[0][0]`	`a[0][1]`	`a[0][2]`
Fila 1	`a[1][0]`	`a[1][1]`	`a[1][2]`

Arreglos Multidimensionales en C

```
int a[2][3] = {{3, 8, 4}, {-3, 6, 1}};
int a[2][3] = {3, 8, 4, -3, 6, 1};
```

	Columna 0	*Columna* 1	*Columna* 2
Fila 0	3	8	4
Fila 1	-3	6	1

Arreglos Multidimensionales en C

```
int a[2][3] = {{3, 8}, {-3, 6, 1}};
```

	Columna 0	*Columna* 1	*Columna* 2
Fila 0	3	8	0
Fila 1	-3	6	1

Arreglos Multidimensionales en C

```
int a[2][3] = {3, 8, 4, -3};
```

	Columna 0	*Columna* 1	*Columna* 2
Fila 0	3	8	4
Fila 1	-3	0	0

Relación entre Punteros y Arreglos en C

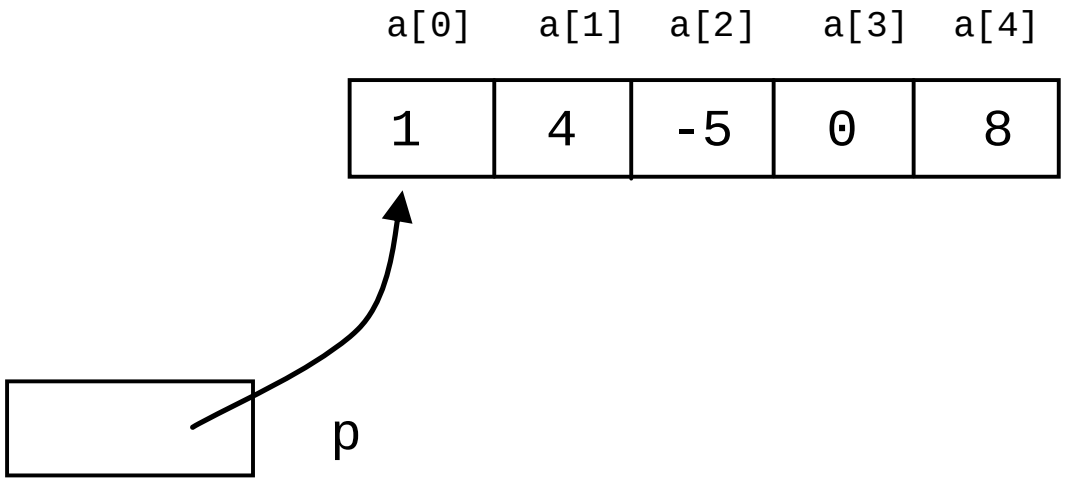
- En C, todas las operaciones que se realizan con arreglos pueden ser realizadas con punteros.
- **En C, la dirección base de un arreglo es equivalente al nombre del arreglo.**

Ejemplo:

```
int b[100];
```

`b` **es equivalente a** `&b[0]`

```
int a[5] = {1, 4, -5, 0, 8};
int *p;
p = a; /*es equivalente a p = &a[0];*/
```

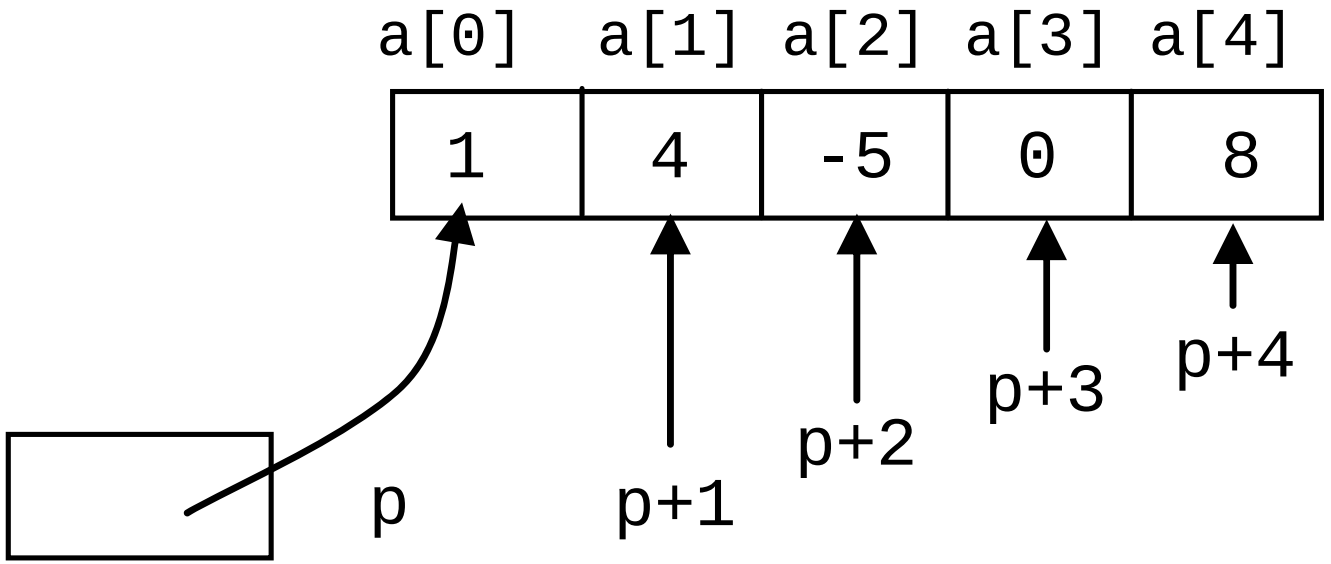



```
printf("%d", a[0]);
printf("%d", *p);
printf("%d", *a);
```

Producirán el mismo efecto

Relación entre Punteros y Arreglos en C
Aritmética de Punteros

- En C, se puede realizar, bajo ciertas condiciones, algunas operaciones aritméticas con punteros :
  - Un puntero puede ser incrementado (++) o decrementado (--).
  - Se le puede sumar o restar un valor entero.
  - Pueden sumarse y restarse punteros entre sí.



p	&a[0]
p+1	&a[1]
p+2	&a[2]
...	...
p+i	**&a[i]**
*p	a[0]
*(p+1)	a[1]
*(p+2)	a[2]
...	...
***(p+i)**	**a[i]**

Ejemplo1:

```
int x[5] = {10, -3, 7, 6, 4};

&x[0]   x            Dirección base del arreglo
&x[2]   (x + 2)      Dirección del 3er elemento
x[0]    *x           1er elemento (10)
x[2]    *(x + 2)     3er elemento (7)
```

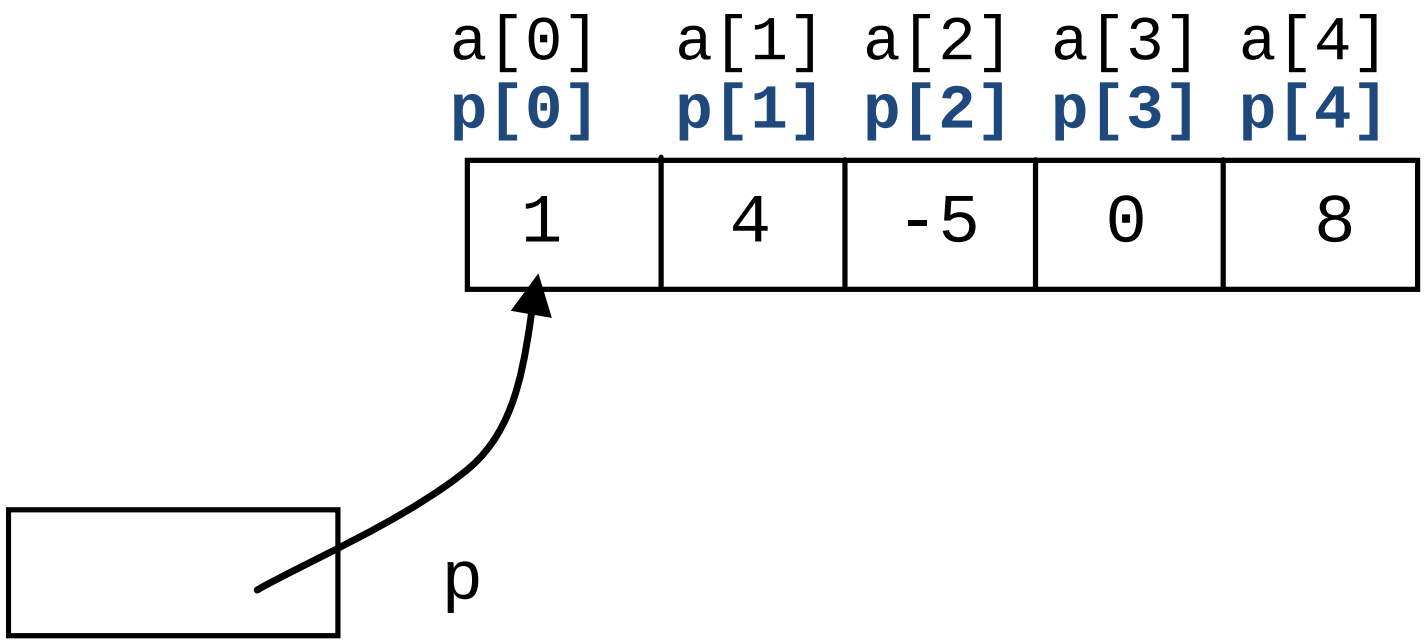
Ejemplo 2:

```
int a[5] = {1, 4, -5, 0, 8};
int *p, *q, i;
p = a;
q = &a[3];
i = q – p;
```

¿Qué valor contendrá la variable `i`?

- Un puntero, cuando apunta a la dirección base del arreglo, puede ser usado con suscriptos:

  `a[i]` podría ser representado como `p[i]`



Pasaje de un Arreglo como Parámetro de una Función en C

Siempre el pasaje de los arreglos en C es por referencia (simulado) ya que lo que se pasa es el valor de la dirección base del arreglo.

```
int ar[33];
```

- En la invocación:

```
modificarArreglo(ar, 33);
```

es equivalente a:

```
modificarArreglo(&ar[0], 33);
```

Pasaje de un Arreglo como Parámetro de una Función en C

Declaración del parámetro formal

```
void modificarArreglo(int b[], int max)
{ … }
```

No hace falta el tamaño, el compilador lo ignora.

Es lo mismo que:

```
void modificarArreglo(int *b, int max)
{ … }
```

Pasaje de un Arreglo como Parámetro de una Función en C

Declaración del prototipo

```
void modificarArreglo(int [], int);
```

O bien

```
void modificarArreglo(int *, int);
```

Ejemplo

```
#include <stdio.h>
#define MAX     5

void muestraArreglo(int [], int);

void modificaArreglo(int [], int);

void modificaUnElemento(int *, int);
```

```
int main()
{
    int a[MAX] = {2, 4, 6, 8, 10};
    int i;

    printf("Los valores del arreglo antes de”
           “ modificarlos son:\n");
    muestraArreglo(a, MAX);
    modificaArreglo(a, MAX);
    printf(“Valores del arreglo despues de llamar”
           “ a modificaArreglo son:\n”);
    muestraArreglo(a, MAX);
    for (i = 0; i < MAX; i++)
        modificaUnElemento(&a[i], i);
    printf(“Valores del arreglo despues de llamar”
           “ a modificaUnElemento son:\n”);
    muestraArreglo(a, MAX);
    return 0;
}
```

```
void muestraArreglo(int b[], int max)
{
     int i;

     for (i = 0; i < max; i++)
         printf("%d ", b[i]);
     printf("\n");
}

void modificaArreglo(int b[], int max)
{
     int i;

     for (i = 0; i < max; i++)
         b[i] = b[i] + 1;
}
```

```
void modificaUnElemento(int *elem, int pos)
{
     *elem = 0;
     printf(“El valor del elemento %d dentro”
          “de modificaUnElemento es: %d\n”,
          pos, *elem);

}
```

Definiciones alternativas equivalentes

```
void modificaArreglo(int b[], int max){
     int i;

     for (i = 0; i < max; i++)
         b[i] = b[i] + 1;
}
void modificaArreglo(int *b, int size){
     int i;

     for (i = 0; i < max; i++)
         b[i] = b[i] + 1;
}
```

```
void modificaArreglo(int *b, int size){
     int i;

     for (i = 0; i < max; i++)
         *(b + i) = *(b + i) + 1;
}

void modificaArreglo(int b[], int size){
     int i;

     for (i = 0; i < max; i++)
         *(b + i) = *(b + i) + 1;
}
```

Ejercicio

```
int a[3] = {5,8,1};
int *p, *q, i;
p = a;
q = &a[1];
i = *p + *q;
*p += *q; /* equiv. *p = *p + *q; */
*(p+1) = 0;
(*(q+1))++;
```

Pasaje de Arreglos Multidimensionales como Parámetros de una Función en C

- Dado que se trata de arreglos de arreglos, no hace falta incluir el tamaño de la 1ra dimensión, pero sí el de las siguientes:

```
int a[2][3];
void f(int x[][3]){
  …
}
```

- Invocación igual que en los unidimensionales

```
f(a);
```

Arreglos de Caracteres y Strings en C

- Usados como estructura soporte en C para los strings (cadenas de caracteres).
- Características particulares:
  1) Inicialización:

```
char s[] = “Programacion I”;
char s[] = {‘P’, ‘r’, ‘o’, ‘g’, ‘r’, ‘a’,
            ‘m’, ‘a’, ‘c’, ‘i’, ‘o’,
            ‘n’,‘ ’,‘I’,‘\0’};
```

  - No todos los arreglos de caracteres representan strings, solo los terminados en ‘\0’.

Arreglos de Caracteres y Strings en C

2) Lectura de una sola vez:

```
char cad[10];
scanf("%s", cad);
```

Responsabilidad del programador asegurarse que el arreglo tenga el tamaño suficiente.

```
scanf("%9s", cad); /* ignora mas
  alla de los 9 caracteres leidos */
```

3) Impresión de una sola vez:

```
printf("%s", cad); /* hasta que encuentra
  el primer carácter '\0' */
```

Strings, Arreglos de Caracteres y Punteros a `char`

```
char s1[] = "hola";
char *s2 = "hola";
```

- `s1` y `s2` almacenan el string "`hola`" en un arreglo de 5 posiciones;
- `s1` y `s2` apuntan a la dirección base del correspondiente arreglo.
- `s1` es un valor constante ya que siempre representa la dirección base del arreglo `s1`, es decir `&s1[0]`.

  `s1` no puede cambiar su valor, por ej., no podemos hacer `s1++`
- Para copiar un string en otro deberemos copiar elemento a elemento usando `strcpy` (en `string.h`).

REGISTROS

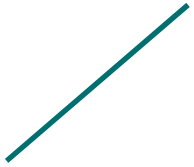
Registros

- Llamados también tuplas, estructuras, records, structs, …
- Estructura heterogénea
- Selector: identificador de campo del registro (explícito)

- Capacidad estática.
- Operaciones: asignación e inspección.
- No hay orden en sus elementos.

Registros en C: Structs

```
struct nombre-registro {
   tipo nombre-campo;
   tipo nombre-campo;
   . . .
}
/*Tipo struct fecha */
struct fecha{
  int anio;
  int mes;
  int dia;
}
```



Los campos de un struct pueden ser de cualquier tipo simple o estructurado

Registros en C: Structs

```
struct fecha{
  int anio;
  int mes;
  int dia;
} varFecha;
```

Declaración del tipo `struct fecha` y de la variable `varFecha`. Reserva memoria para la variable.

```
struct fecha fechaNacimiento,
             fechaCasamiento;
```

Declaración de las variables `fechaNacimiento` y `fechaCasamiento` de tipo `struct fecha`. Cada declaración reserva memoria.

Registros en C: Structs

- No es obligatorio darle un nombre al struct:

```
struct {
  int anio;
  int mes;
  int dia;
} varFecha;
```

  sin embargo, no lo vamos a poder usar en otra declaración de variable.

- Siempre debe estar presente al menos el nombre de la estructura o el nombre de la variable.

Operaciones con structs

- Es posible asignar un struct a otro, siempre que sean del mismo tipo, pero no se pueden comparar:

```
struct fecha{
  int anio;
  int mes;
  int dia;
} fechaNacimiento = { 1988, 10, 5};
struct fecha copiaFecha;
copiaFecha = fechaNacimiento;    /* OK */
if (copiaFecha == fechaNacimiento)
   printf("Iguales");           /* Error */
```

Operaciones con structs

- Obtener la dirección de un struct usando el operador de dirección (`&`).
- Acceder a sus campos por medio de su selector.
- Usar el operador `sizeof` para determinar su tamaño en bytes.
- Ser pasados como parámetros de una función y pueden también ser devueltos como resultado.

Inicialización de los structs

- Una variable de tipo struct también puede inicializarse al momento de la declaración

```
struct {
  int anio;
  int mes;
  int dia;
} fechaNacimiento = { 1988, 10, 5}
```

- Si faltan valores, inicializa en 0.
- Si no se hace al momento de la declaración, entonces hay que hacerlo campo a campo.

Acceso a los campos de un struct

```
struct {
  int anio;
  int mes;
  int dia;
} fechaNacimiento = {1988, 10, 5};

fechaNacimiento.dia = 29;
fechaNacimiento.mes = 6;
printf("La fecha de nacimiento de Juana es
  el %d/%d/%d\n", fechaNacimiento.dia,
  fechaNacimiento.mes,fechaNacimiento.anio);
```

Pasaje de un struct como parámetro de una función

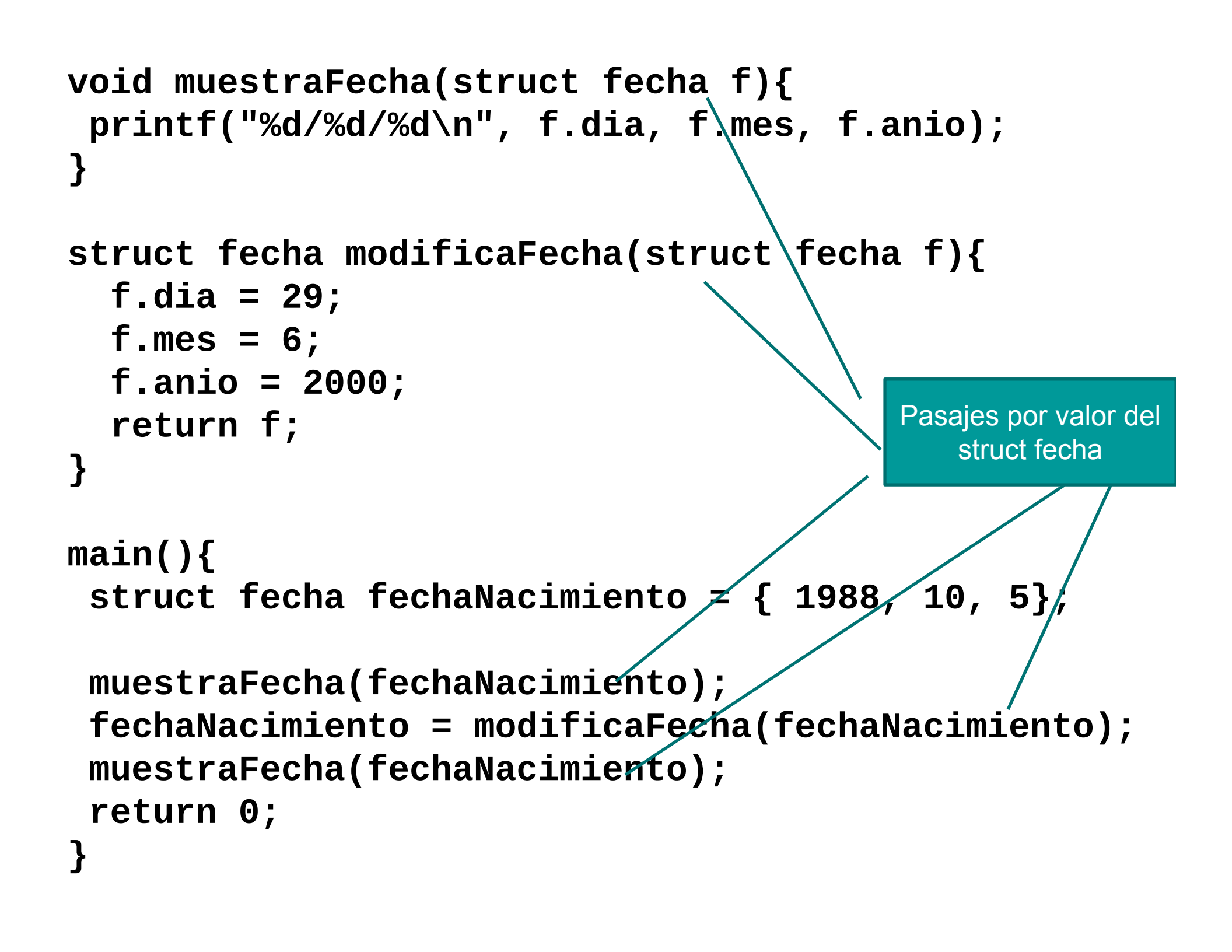
- Al igual que cualquier otra variable en C, las variables de tipo struct son pasadas por valor.

- Para pasar un registro por dirección deberemos simular el pasaje por dirección (declarando el parámetro formal de tipo puntero y anteponiendo el operador de dirección & al parámetro actual).

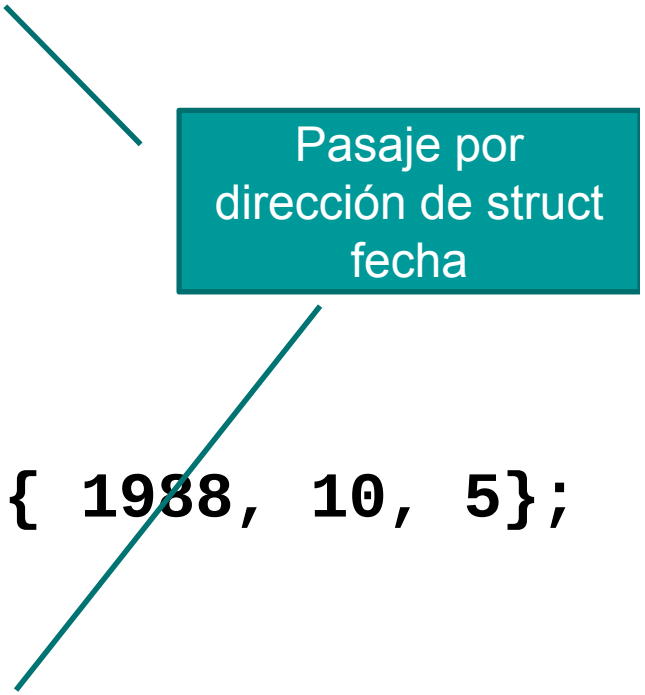
Ejemplo de pasaje de un struct como parámetro de una función

```
#include <stdio.h>

/* declaracion global del tipo struct
  fecha*/
struct fecha{
  int anio;
  int mes;
  int dia;
};
```

```
void muestraFecha(struct fecha f){
 printf("%d/%d/%d\n", f.dia, f.mes, f.anio);
}

struct fecha modificaFecha(struct fecha f){
  f.dia = 29;
  f.mes = 6;
  f.anio = 2000;
  return f;
}

main(){
 struct fecha fechaNacimiento = { 1988, 10, 5};

 muestraFecha(fechaNacimiento);
 fechaNacimiento = modificaFecha(fechaNacimiento);
 muestraFecha(fechaNacimiento);
 return 0;
}
```



```
void muestraFecha(struct fecha f) {
  printf("%d/%d/%d\n", f.dia, f.mes, f.anio);
}

void modificaFecha(struct fecha *f) {
  (*f).dia = 29;
  (*f).mes = 6;
  (*f).anio = 2000;
}

main(){
  struct fecha fechaNacimiento = { 1988, 10, 5};

  muestraFecha(fechaNacimiento);
  modificaFecha(&fechaNacimiento);
  muestraFecha(fechaNacimiento);
  return 0;
}
```



Uso de `typedef`

- Permite crear sinónimos o alias para tipos de datos ya definidos:

```
struct fecha{
  int anio;
  int mes;
  int dia;
};
typedef struct fecha Fecha ;

void muestraFecha(Fecha f);
void modificaFecha(Fecha *f);

Fecha fecNac;
```

Uso de `typedef`

```
typedef struct {
  int anio;
  int mes;
  int dia;
} Fecha;
void muestraFecha(Fecha f);
void modificaFecha(Fecha *f);
Fecha fecNac;
```

Punteros a structs

```
typedef struct triangulo {
  float base;
  float altura;
} Triangulo;
Triangulo t = {5.0, 10.0};
Triangulo *pTriang;
pTriang = &t;
(*pTriang).base = 6.0;
(*pTriang).altura = 5.8;
pTriang->base = pTriang->base / 2;
```