

---

# Programación I

## Teoría II

---

<http://proguno.unsl.edu.ar>  
[proguno@unsl.edu.ar](mailto:proguno@unsl.edu.ar)


---

# **MODULARIDAD FUNCIONES EN C**

---

---

# Modularidad

- Principio para resolución de problemas:
    - “Dividir para reinar”
  - Modularidad
  - Módulo
    - Función
    - Procedimiento
    - Subrutina
    - Subprograma
    - ...
- 

---

# Modularidad

## ■ Ventajas

- ❑ Re-uso de código
- ❑ Abstracción: concentrarse en aspectos esenciales y no accidentales.
- ❑ Visión de caja negra (encapsulamiento).

## ■ Partes de un módulo

- ❑ Una *Definición*
  - ❑ Una o más *Invocaciones o llamadas*
-

---

# Las funciones en C

## 1. Funciones *pre-definidas* agrupadas en bibliotecas

- Ejemplos:

- Manejo de strings (`string.h`)
- Funciones matemáticas (`math.h`)
- I/O (por ej. `scanf`, `printf`) (`stdio.h`)
- ...

## 2. Funciones *definidas por el usuario*

- Una *Definición* de la función + [declaración del prototipo]
  - Una ó más *invocación(es)*
-

---

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

```
int potencia(int, int); /* prototipo */
```

```
main()
```

```
{
```

```
    int i ;
```

```
    for (i = 0; i <= 10 ; i++)
```

```
        printf("%d %d %d \n", i,
```

```
            potencia(2, i), potencia(3, i));
```

```
    return 0;
```

```
}
```

---

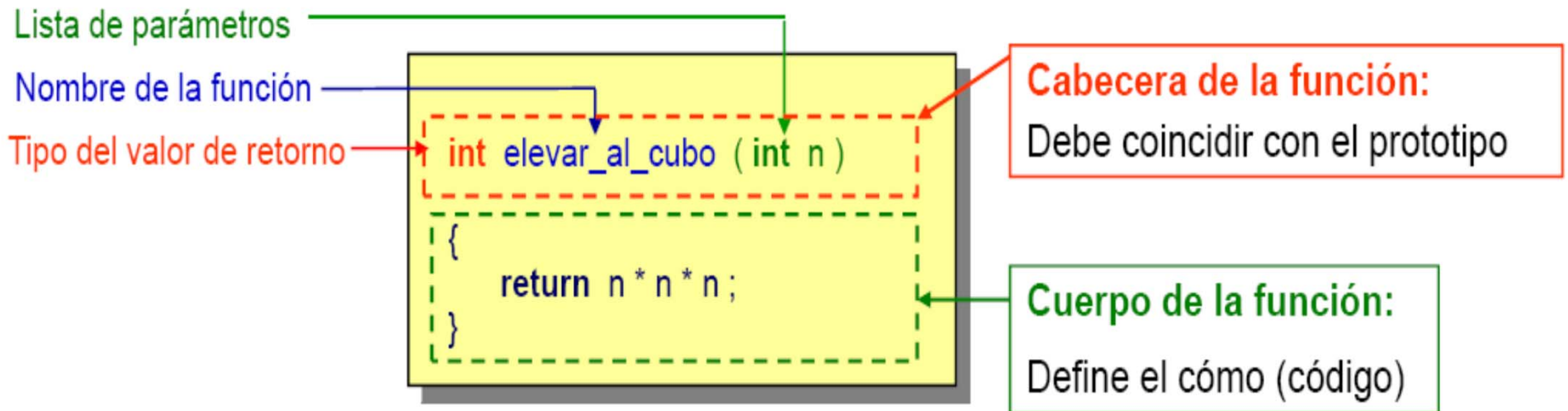
---

```
/* Función que eleva la base n a la
   potencia m (m >= 0)*/
int potencia(int n, int m)
{
    int i, p;

    p = 1;
    for (i = 0; i < m; i++)
        p = p * n;
    return p;
}
```

---

# Definición de una función en C





---

# Definición de una función en C

## *Tipo del Valor de Retorno*

- Cuando la función no retorna nada se coloca `void`

```
void imprimeNumerosDesdeHasta(int x, int y){  
    for (int i = x; i <= y; i++)  
        printf("%d\n", i);  
}
```

- Si se omite *tipo-de-retorno* se asume `int`

```
potencia(int n, int m){  
    int i, p;  
    p = 1;  
    for (i = 1; i <= m; i++) p = p * n;  
    return p;  
}
```

---

---

# Definición de una función en C

*Lista de parámetros formales (opcionales)*

- Si la función no tiene parámetros formales se coloca `void`

```
int flip(void){  
    /* genera un número random */  
    ...  
}
```

---

---

# Definición de una función en C

*Valor retornado en el cuerpo de la función*

- Se usa sentencia `return` para devolver valor del tipo de retorno de la función.
  - Puede no devolver nada (la función retorna tipo `void`).
    - `return;`
    - Sin sentencia `return`, retorna cuando encuentra `}`
-

---

# Definición de una función en C

## *Definición del prototipo de la función*

- ❑ Es idéntico al encabezado de la definición de la función, aunque los identificadores de los parámetros pueden ser distintos o incluso omitirse.
- ❑ Es usado por el compilador para controlar las invocaciones a la función.
- ❑ Ejemplos:

```
int potencia(int, int);
```

```
int potencia(int n, int m);
```

```
int potencia(int a, int b);
```

---

---

# Invocación a una función en C

*identificador(lista-opcional-de-parametros-actuales)*

## ■ Ejemplos:

```
void cambiaChar(int, char); /* prototipo char a int
                             */
int cambiar(void); /* prototipo sin parámetro */
char minuscula_mayus(char); /* prototipo */
...
char c = 'a';
/* las siguientes invocaciones son correctas?*/
int i = cambiar();
cambiar();
printf("%f", minuscula_mayus(c));
cambiaChar(10, '*');
```

---

---

# Alcance o ámbito de los identificadores

- El *alcance* o *ámbito* de un identificador determina en qué parte del programa el identificador está definido y puede usarse.
  - *Reglas de ámbito o alcance en C:*
    1. Alcance de archivo;
    2. Alcance de bloque;
    3. Alcance de prototipo;
    4. Alcance de función.
-

---

# Alcance de Archivo

- Tienen alcance de archivo los identificadores declarados por fuera de una función.
  - Son accesibles en el archivo desde el punto de su declaración y hasta el final del archivo.
  - Conocidos como *globales*
    - Variables globales;
    - Definiciones de funciones y de prototipos de funciones.
-

```
#include <stdio.h>

int global = 3; /* variable global */

void cambiaGlobal()
{
    global = 5;
}

int main()
{
    printf("%d\n", global);
    cambiaGlobal();
    printf("%d\n", global);
    return 0;
}
```



---

# Alcance de Bloque

- Tienen alcance de bloque los identificadores declarados dentro de un bloque o de una lista de parámetros formales.
  - Son accesibles sólo desde el punto de su declaración o definición y hasta el final del bloque que los contiene.
  - Conocidos como identificadores *locales*
    - Variables locales.
    - En C no hay funciones anidadas, en consecuencia no hay funciones locales.
-

```
#include <stdio.h>

int cubo(int); /* prototipo */
int variable_global = 1;

void main (void)
{
    int c = cubo(5);
    variable_global = 3;
    printf("5 al cubo=%d, 3 al cubo=%d \n", c, cubo(3));
    resultado = 9; /* error !!! */
}

int cubo(int n)
{
    int resultado = n * n * n;
    variable_global = 2;
    c = 9; /* error !!!*/
    return resultado;
}
```

```
#include <stdio.h>
int main()
{
    int j,i;
    printf("ingrese el valor de i:");
    scanf("%d",&i);
    printf("ingrese el valor de j:");
    scanf("%d",&j);

    if (j) {
        int j = 0;
        printf("i = %d, j = %d\n", i, j);
    }
    else {
        int i = 0;
        printf("i = %d, j = %d\n", i, j);
    }
    printf("i = %d, j = %d\n", i, j);
    return 0;
}
```

---

# Alcance de Prototipo

- Tienen alcance de prototipo los identificadores usados dentro de una lista de parámetros en la declaración de un prototipo.
  - Su ámbito es sólo la lista de parámetros, no interfiriendo con las variables de otros puntos del programa.
-

```
#include <stdio.h>
/* variable x con alcance de archivo */
int x = 4;

/* variable x con alcance de prototipo*/
long mifunc(int x, long y);

int main(void){
    /* variable x con alcance de bloque */
    int x = 5;
    printf("%d\n",x);
}
```

---

# Alcance de Función

- Los únicos identificadores con este alcance son las *etiquetas* (un identificador seguido por :).
  - Su declaración es implícita (en el lugar que se las usa por primera vez).
  - Son accesibles o conocidas en toda la función en donde aparecen.
  - Usadas en conjunción con la sentencia `goto`.
  - Nosotros no usaremos `goto` ni etiquetas.
-

---

# Tiempo de Vida o Permanencia de los Identificadores

- Período durante el cual un identificador existe en la memoria de la computadora.
  - No necesariamente coincide con su alcance.
    - Identificadores globales
      - *Estática* (desde que se inicia la ejecución del programa y hasta que finaliza).
    - Identificadores locales
      - *Automática* (desde que se inicia la ejecución del bloque y hasta que termina).
      - Se puede forzar la permanencia estática usando el modificador `static` cuando se declara el identificador
-

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

```
int f(void) {
```

```
    static int x = 0; /* local y estática */
```

```
    x++;
```

```
    return x;
```

```
}
```

```
int main(void) {
```

```
    int j;
```

```
    for (j = 0; j < 5; j++) {
```

```
        printf("Valor de f(): %d\n", f());
```

```
    }
```

```
    return 0;
```

```
}
```



---

# **EL TIPO PUNTERO**

---

---

# Punteros en C

- Un *puntero* es un apuntador a un lugar de la memoria que puede almacenar datos de un determinado tipo.
- Una variable de tipo puntero contiene una dirección en donde se encuentra almacenado un dato de un determinado tipo.
- Ejemplos de declaraciones en C:

```
char *p; /* puntero a char */  
int *ptr; /* puntero a int */  
float *punt; /* puntero a float*/
```

---

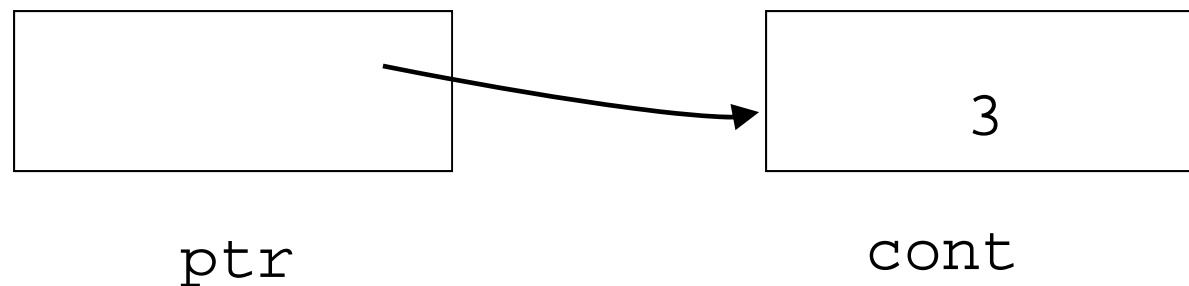
---

# Punteros en C

## Referencia directa vs. Referencia indirecta

```
int cont, *ptr;
```

- `cont` referencia de forma directa al valor 3
- `ptr` lo referencia de forma indirecta.

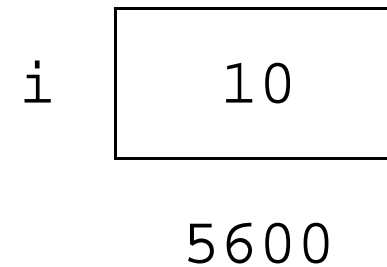


---

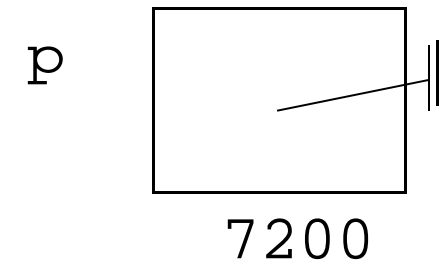
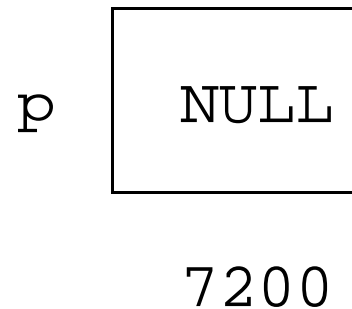
# Operaciones con Punteros en C

## Inicializaciones

```
int i = 10;
```



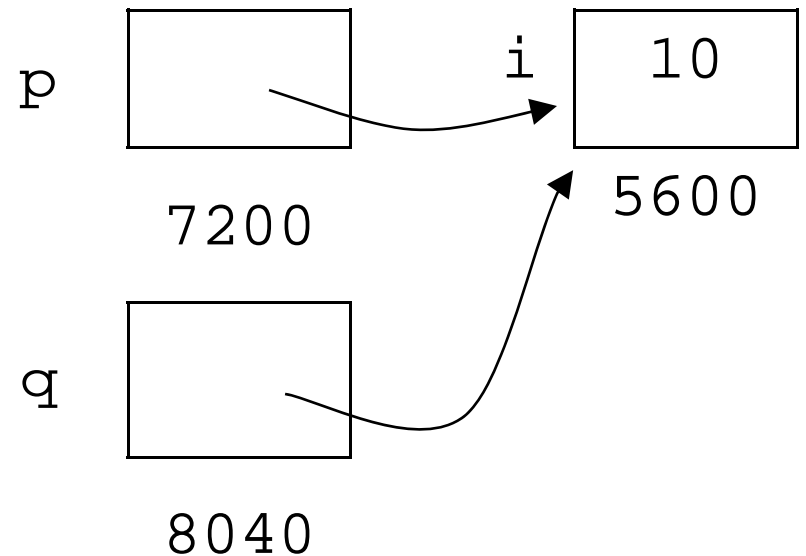
```
int *p = NULL;  
int *p = 0;
```



# Operaciones con Punteros en C

## Operador de Dirección &

```
int i = 10;  
int *p = &i;  
int *q = p;
```



---

# Operador de Desreferenciación o de Indirección \*

- Permite acceder al valor apuntado por una variable de tipo puntero.

```
printf("valor apuntado por p %d\n", *p);  
*p = *p * 5;  
printf("valor apuntado por p %d\n", *p);  
printf("valor apuntado por q %d\n", *q);  
printf("valor de i %d\n", i);
```

---

---

# Pasaje de Parámetros

- Los módulos se comunican entre sí por medio de los datos que intercambian a través del uso de parámetros.
  - Parámetros
    - Formales (en la definición).
    - Actuales o reales (en la invocación).
  - Pasajes de parámetros
    - Por dirección o referencia o variable.
    - Por valor o constante.
-

---

# Pasaje de Parámetros

Pasaje por valor	Pasaje por referencia
Se pasa una copia del dato, pero no el dato mismo.	Se pasa la dirección de memoria en la que se encuentran los datos.
El módulo puede modificar la copia, pero nunca el dato original.	El módulo accede a los datos libremente para leerlos y/o escribirlos.
Seguro pero ineficiente cuando hay que pasar datos estructurados (grandes).	Inseguro y obliga al programador a ser muy cuidadoso, pero es una forma muy eficiente de pasar estructuras de datos grandes y complejas.

---



---

# Pasaje de Parámetros en C

- En C, sólo hay pasaje por valor pero...
  - Se puede simular el pasaje por referencia usando tipos **punteros** y pasando la dirección del parámetro actual...
-

---

# Pasaje de Parámetros en C

- En el siguiente ejemplo, la función `intercambia`, dadas dos variables enteras, debería intercambiar sus valores...
-

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

```
void intercambia(float x, float y){  
    float temp;  
    temp = x;  
    x = y;  
    y = temp;  
}
```

```
main(){  
    float a = 1.0;  
    float b = 2.0;  
  
    printf("a = %f, b = %f\n", a, b);  
    intercambia(a, b);  
    printf("a = %f, b = %f\n", a, b);  
}
```

---

# Pasaje de Parámetros en C

- La función `intercambia`, ¿hizo lo que se pretendía?
  - ¿Por qué?
  - ¿Cómo resolvemos el problema?
-

---

¿Cómo solucionamos este problema?

## Simulando el Pasaje por Dirección

- Pasamos la dirección del o los parámetros que queremos que se modifiquen en la función, por lo tanto esos parámetros tendremos que declararlos de tipo puntero ya que lo que pasaremos es una dirección.
  - Dentro de la función podremos modificar o acceder a los valores apuntados usando el operador de desreferenciación \*.
-

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

```
void intercambia(float *x, float *y){  
    float temp;  
    temp = *x;  
    *x = *y;  
    *y = temp;  
}
```

```
main(){  
    float a = 1.0;  
    float b = 2.0;  
  
    printf("a = %f, b = %f\n", a, b);  
    intercambia(&a, &b);  
    printf("a = %f, b = %f\n", a, b);  
}
```