



FUNCIONES EN C

FUNCIONES

- Los módulos en C se llaman **funciones**.
- Hemos estado utilizando funciones de la biblioteca estandar “stdio.h” como por ejemplo **printf** y **scanf**.
- Comenzaremos viendo algunas funciones de la biblioteca matemática y luego detallaremos la manera de definir funciones en C.
- En C, las funciones sólo reciben parámetros por valor



FUNCIONES MATEMÁTICAS (MATH.H)

Función	Descripción	Ejemplo
<code>sqrt(x)</code>	Raíz cuadrada de x	<code>sqrt(900.0)</code> es 30.0
<code>exp(x)</code>	Función exponencial e^x	<code>exp(1.0)</code> es 2.718282 <code>exp(2.0)</code> es 7.389056
<code>log(x)</code>	Logaritmo natural de x (base e)	<code>log(2.718282)</code> es 1.0 <code>log(7.389056)</code> es 2.0
<code>log10(x)</code>	Logaritmo de x base 10	<code>log10(1.0)</code> es 0.0 <code>log10(10.0)</code> es 1.0



FUNCIONES MATEMÁTICAS (MATH.H)

Función	Descripción	Ejemplo
<code>fabs(x)</code>	Valor absoluto de x	Si $x > 0$, <code>fabs(x)</code> es x Si $x = 0$, <code>fabs(x)</code> es 0 Si $x < 0$, <code>fabs(x)</code> es -x
<code>ceil(x)</code>	Redondea x al entero menor que no sea inferior a x	<code>ceil(9.2)</code> es 10.0 <code>ceil(-9.8)</code> es -9.0
<code>floor(x)</code>	Redondea x al entero más grande no mayor que x	<code>floor(9.2)</code> es 9.0 <code>floor(-9.8)</code> es -10.0
<code>pow(x,y)</code>	x elevado a la potencia y (x^y)	<code>pow(2,7)</code> es 128.0 <code>pow(9, 0.5)</code> es 3.0



FUNCIONES MATEMÁTICAS (MATH.H)

Función	Descripción	Ejemplo
fmod(x)	Residuo de x/y como un número de punto flotante	fmod(13.657, 2.333) es 1.992 $13.657 = 5 * 2.333 + \mathbf{1.992}$
sin(x)	Seno de x (x en radianes)	sin(0.0) es 0.0
cos(x)	Coseno de x (x en radianes)	cos(0.0) es 1.0
tan(x)	Tangente de x (x en radianes)	tan(0.0) es 0.0



FUNCIONES EN C

- **Sintaxis**

```
TipoValorRetornado NombreDeLaFuncion(parámetros)
{ declaraciones locales
  Instrucciones de la función
}
```

- **Ejemplo**

```
int Cuadrado ( int nro)
{ int resultado;
  resultado = nro * nro;
  return (resultado) ;
}
```

```
int Cuadrado ( int nro)
{
  return (nro * nro) ;
}
```



```
/* Función cuadrado definida  
   por el programador */  
#include <stdio.h>
```

```
int cuadrado(int);
```



Prototipo de la función

```
int main()  
{ int x;
```

Invocación



```
    for (x=1; x<=10; x++)  
        printf("%d    ", cuadrado(x));
```

```
    return 0;
```

```
}
```

Definición de la función

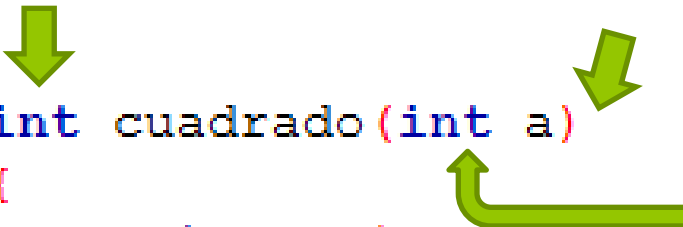
```
/* Definición de la función */  
int cuadrado(int a)  
{  
    return a*a;  
}
```



DEFINICIÓN DE UNA FUNCIÓN

Si se omite el tipo del valor a devolver se asumirá **int**

Note que no lleva ; al cerrar el paréntesis



```
int cuadrado(int a)
{
    return a*a;
}
```

Cada parámetro debe ir precedido por el nombre del tipo. Si se omite se asume **int**

- También pudo haberse codificado así pero **NO cumple con ANSI C**

```
int cuadrado(a)
int a
{    return a*a
}
```

```
cuadrado(a)
int a
{    return a*a
}
```



```
/* Función cuadrado definida  
   por el programador */  
#include <stdio.h>
```

```
int cuadrado(int);
```

 **Prototipo de la función**

- Permite al compilador hacer validaciones referidas a los tipos, cantidad y orden de los parámetros y al tipo de valor retornado.
- También podíamos poner
int cuadrado(int a)
pero el compilador ignora los nombres de los parámetros.
- El prototipo de función se caracteriza por la **coerción de argumentos** ya que fuerza su conversión al tipo apropiado.
- Si el prototipo de la función se omite se tomará la primera invocación como prototipo. Esto puede llevar a errores.

COERCIÓN DE TIPOS

```
/* Función cuadrado definida  
   por el programador */  
#include <stdio.h>
```

```
double cuadrado(double);
```

Qué pasa si
comentamos
esta línea?

```
int main()  
{ int x;
```

```
    for (x=1; x<=10; x++)  
        printf("%3.0f  \n ", cuadrado(x));
```

```
    return 0;
```

```
}
```

y si cambiamos por %3d ?

```
double cuadrado(double a)  
{  
    return a*a;  
}
```



RETORNO DE LA FUNCIÓN

- Hay tres formas de regresar al punto desde el cual se hizo la invocación de la función:
 - Si la función no retorna nada, el control sólo se devuelve cuando se llega a la llave derecha que termina la función.
 - Cuando se ejecuta la instrucción
return;
 - Si la función devuelve un resultado, la instrucción
return expresion;
devuelve el valor de **expresion** al punto de llamada.



RETORNO DE LA FUNCIÓN

- Hay tres formas de regresar al punto desde el cual se hizo la invocación de la función:
 - Si la función no retorna nada, el control sólo se devuelve cuando se llega a la llave derecha que termina la función.
 - Cuando se ejecuta la instrucción
return;
 - Si la función devuelve un resultado, la instrucción
return expresion;
devuelve el valor de **expresion** al punto de llamada.



RETORNO DE LA FUNCIÓN

Qué
devuelve?

```
#include <stdio.h>
int main()
{
    int a,b;

    printf("Ingrese dos números enteros : ");
    scanf("%d %d", &a, &b);

    printf("El mayor es %d\n", Mayor(a,b));
    return 0;
}

int Mayor(int nro1, int nro2)
{
    return (nro1);
    if (nro2>nro1)
        return(nro2);
}
```



TIPO VOID

Void es un tipo de dato que representa que no hay valor.

```
#include <stdio.h>
void saludo(void);
void main()
{
    saludo();
    saludo();
}

void saludo(void)
{
    printf("Bienvenido al ");
    printf("curso de Taller de ");
    printf("Lenguajes 1!\n\n\n");
}
```

No cumple con el estándar. Se espera que la función **main** siempre retorne un entero



EJERCICIO

- Escriba una función que reciba tres números y retorne el valor mayor.
- Utilice la función anterior para desarrollar un programa que lea tres números enteros de teclado e indique cual fue el mayor valor ingresado.
- Puede utilizar la misma función para hallar el máximo de tres números sin importar si tienen decimales o no?



PASAJE DE PARÁMETROS

- El lenguaje C sólo posee pasaje de parámetros por copia o por valor. Por lo tanto no es posible cambiar el valor de un parámetro desde una función.
- ¿Cómo hacer entonces para permitir que un función devuelva algo a través de su lista de parámetros?

utilizando punteros





PUNTEROS EN C

PUNTEROS

- Permiten simular el pasaje de parámetros por referencia.
- Permiten crear y manipular estructuras de datos dinámicas.
- Su manejo es de fundamental importancia para poder programar en C.



PUNTEROS

- Un puntero es una variable que contiene una dirección de memoria.
- Por lo general, una variable contiene un valor y un puntero a ella contiene la dirección de dicha variable.
- Es decir que la variable se refiere **directamente** a un valor mientras que el puntero lo hace **indirectamente**.

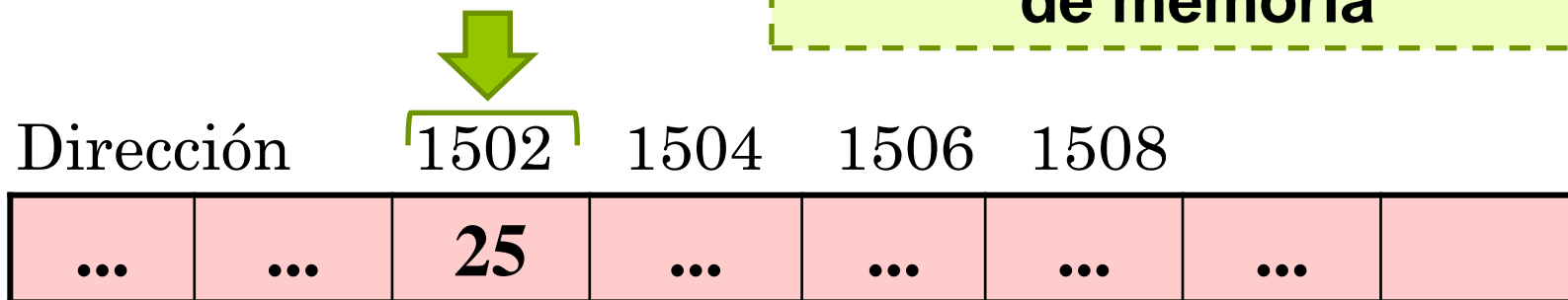


DIRECCIÓN Y CONTENIDO DE MEMORIA

- Una dirección de memoria y su contenido no es lo mismo.

```
int x = 25;
```

Un **puntero** es una variable que contiene una **dirección de memoria**



La **dirección** de la variable x es 1502

El **contenido** de la variable x es 25



DECLARACIÓN DE PUNTEROS

○ Ejemplo

```
int *countPtr, count;
```



Puntero a un
entero



Es un entero
(NO un puntero)

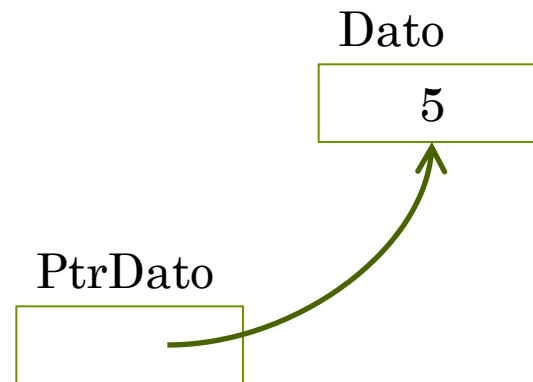
- El * no se aplica a todos los nombres de variables de una declaración. Cada puntero debe llevar su nombre precedido por *.



OPERADORES DE PUNTEROS

- El operador & u *operador de dirección*, es un operador unario que retorna la dirección de su operando.
- **Ejemplo**

```
int Dato = 5;  
int *PtrDato;  
  
PtrDato = &Dato;
```



OPERADORES DE PUNTEROS

- El operador *, también llamado *operador de dirección*, retorna el valor del objeto hacia el cual apunta su operando.
- **Ejemplo**

```
int Dato = 5, *PtrDato;
```

```
PtrDato = &Dato;
```

```
printf("%d\n", *PtrDato);
```



Imprime 5



PUNTEROS

- El puntero debe contener una dirección a un elemento del mismo tipo que la variable apuntada.

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

```
int main()
```

```
{  int *ptr;  
    int dato=30;
```



Declara un puntero a un entero

```
ptr = &dato;
```

```
*ptr = 50;
```

```
printf("Dato = %d\n", dato);
```

```
return 0;
```

```
}
```



EjemploPtr.c


PUNTEROS

- El puntero debe contener una dirección a un elemento del mismo tipo que la variable apuntada.

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

```
int main()
```

```
{  int *ptr;  
    int dato=30;
```

```
    ptr = &dato;   
    *ptr = 50;
```

& es el **operador de dirección**: permite obtener la dirección de memoria de la variable que le sigue

```
printf("Dato = %d\n", dato);
```

```
return 0;
```

```
}
```

PUNTEROS

- El puntero debe contener una dirección a un elemento del mismo tipo que la variable apuntada.

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

```
int main()
```

```
{  int *ptr;  
    int dato=30;
```



```
ptr = &dato;
```

```
*ptr = 50;
```



```
printf("Dato = %d\n", dato);
```

```
return 0;
```

```
}
```

No hay que confundir el *
que aparece en la
declaración

con

el **operador de
indirección**



PUNTEROS

- El puntero debe contener una dirección a un elemento del mismo tipo que la variable apuntada.

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

```
int main()
```

```
{  int *ptr;  
    int dato=30;
```



```
    ptr = &dato;  
    *ptr = 50;
```

```
    printf("Dato = %d\n", dato);
```

```
    return 0;
```

```
}
```

Cámbielo por
float * ptr

Ejecute y observe el
resultado obtenido

EjemploPtr.c

QUÉ IMPRIME?

```
#include <stdio.h>
int main(void) {
    int a, b, c, *p1, *p2;

    p1 = &a;
    *p1 = 1;
    p2 = &b;
    *p2 = 2;
    p1 = p2;
    *p1 = 0;
    p2 = &c;
    *p2 = 3;
    printf("%d %d %d\n", a, b, c);

    return 0;
}
```



VISUALIZANDO EL VALOR DE UN PUNTERO

- Puede utilizarse printf con la especificación de conversión **%p** para visualizar el valor de una variable puntero en forma de entero hexadecimal.
- **Ejemplo**

```
int Dato = 5, *PtrDato;
```

```
PtrDato = &Dato;
```

```
printf("%p\n", PtrDato);
```



Qué imprime?



VISUALIZANDO EL VALOR DE UN PUNTERO

- Puede utilizarse printf con la especificación de conversión **%p** para visualizar el valor de una variable puntero en forma de entero hexadecimal.
- **Ejemplo**

```
int Dato = 5, *PtrDato;
```

```
PtrDato = &Dato;
```

```
printf("%p\n", PtrDato);
```



0028FF1C



```
/* Operadores & y * */
```

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

```
int main()
```

```
{ int Dato = 5, *PtrDato;
```

```
PtrDato = &Dato;
```

```
printf("\n La direccion de Dato es %p\n"
```

```
      " El valor de PtrDato es %p\n\n",
```

```
      &Dato, PtrDato);
```

```
printf(" La valor de Dato es %d\n"
```

```
      " El valor de *PtrDato es %d\n\n",
```

```
      Dato, *PtrDato);
```

```
printf(" Note la relacion entre * y & \n"
```

```
      " &*PtrDato = %p\n *&PtrDato = %p\n",
```

```
      &*PtrDato, *&PtrDato);
```

```
return 0;
```

```
}
```

Que imprime?



SALIDA DEL PROGRAMA ANTERIOR

```
La direccion de Dato es 0028FF1C  
El valor de PtrDato es 0028FF1C
```

```
La valor de Dato es 5  
El valor de *PtrDato es 5
```

```
Note la relacion entre * y &  
&*PtrDato = 0028FF1C  
*&PtrDato = 0028FF1C
```



PUNTEROS

- Las **direcciones de memoria** dependen de la arquitectura de la computadora y de la gestión que el sistema operativo haga de ella.
- Desde C no es posible indicar numéricamente una dirección de memoria para guardar información (esto se hace a través de funciones específicas).
- Utilizamos punteros para acceder a la información a través de su dirección de memoria.



INICIALIZACIÓN DE PUNTEROS

- Los punteros deben ser inicializados.
- Utilice el identificador **NULL** (definido en `<stdio.h>`) para indicar que el puntero no apunta a nada.
- El **0** es el único valor entero que puede asignarse directamente a un puntero y es equivalente a **NULL**.
- Cuando se asigna **0** a un puntero se realiza un casting previo automático al tipo apropiado.



EJEMPLO

```
#include <stdio.h>
int main()
{
    int *ptr1 = 45637325; ←
    int *ptr2 = 0;
    int *ptr3 = NULL;

    return 0;
}
```

No es posible asignarle un valor fijo a un puntero. No es posible saber si es una posición válida.



EJEMPLO

```
#include <stdio.h>
int main()
{
    int *ptr1 = 45637325;

    int *ptr2 = 0;    ←
    int *ptr3 = NULL;

    return 0;
}
```

El 0 es el único valor que puede asignarse a un puntero.
La conversión a (int *) es automática.



EJEMPLO

```
#include <stdio.h>
int main()
{
    int *ptr1 = 45637325;

    int *ptr2 = 0;

    int *ptr3 = NULL;

    return 0;
}
```

NULL equivale a 0 y está
definido en <stdio.h>





PASAJE DE PARÁMETROS POR REFERENCIA

PASAJE DE PARÁMETROS POR REFERENCIA

- Vimos que en C los parámetros de las funciones siempre se pasan por valor.
- Para simular el pasaje de parámetro por referencia se utiliza la dirección de la variable, es decir, que lo que se envía es un puntero a su valor.
- El puntero es un parámetro sólo de entrada que permite modificar el valor de la variable a la que apunta.



```
/* parámetro por referencia */
```

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

```
void cuadrado(int *);
```

```
int main()
```

```
{ int a = 5;
```

```
printf("Valor original = %d\n", a);
```

```
cuadrado(&a);
```



Envía la dirección de la variable (un puntero)

```
printf("Valor al cuadrado = %d\n", a);
```

```
return 0;
```

Recibe un puntero a un entero

```
}
```

```
void cuadrado(int * nro)
```

```
{
```

```
    *nro = *nro * *nro;
```

```
}
```

Valor de la variable apuntada por **nro**

¿QUÉ IMPRIME?

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

```
void duplicar( int nro );
```

```
int main()
```

```
{
```

```
    int nro = 3;
```

```
    printf("nro = %d\n", nro);
```

```
    duplicar(nro);
```

```
    printf("nro retornado =  %d\n", nro);
```

```
    return 0;
```

```
}
```

```
void duplicar( int nro ){
```

```
    nro = 2 * nro;
```

```
    printf("nro dentro de la funcion = %d\n", nro);
```

```
}
```

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

```
void duplicar( int * );
```

```
int main()
```

```
{
```

```
    int nro = 3;
```

```
    printf("nro = %d\n", nro);
```

```
    duplicar(&nro);
```

```
    printf("nro retornado =  %d\n", nro);
```

```
    return 0;
```

```
}
```

```
void duplicar( int * DirNro ){
```

```
    *DirNro = 2 * (*DirNro);
```

```
    printf("nro dentro de la funcion = %d\n", (*DirNro));
```

```
}
```

EJERCICIO

- Escriba una función que reciba dos números enteros y los devuelva ordenados. Es decir que en el primer parámetro debe devolver el menor valor y en el 2do. el mayor.
- Utilice la función anterior para leer pares de valores e imprimirlos ordenados.



RECURSIÓN

- Al igual que Pascal, C soporta la definición de funciones recursivas.
- Recuerde que el objetivo de este tipo de funciones es reducir la complejidad del problema.
- Se utiliza un caso base no recursivo y en cada llamada se busca reducir el tamaño de la entrada de manera de cercarse a dicho caso base.
- Las soluciones recursivas si bien facilitan la escritura de la solución ocupan más memoria que las soluciones iterativas.



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

```
unsigned int Factorial(unsigned int);
```

```
int main()
```

```
{  
    printf("El factorial de 8 es");  
    printf(" %u\n", Factorial(8));  
  
    return 0;  
}
```

```
unsigned int Factorial(unsigned int nro)
```

```
{  
    if ((nro==0) || (nro==1))  
        return 1;  
    else return(nro * Factorial(nro -1));  
}
```



EJERCICIO

- Escriba una función en C que dado un número entero retorne un unsigned con dicho número representado en binario.

Ej: Recibe el número **27** y devuelve el número **11011**

- Use la función anterior para imprimir la conversión a binario de los múltiplos de 9 menores a 127.
 - Utilice una función iterativa
 - Utilice una función recursiva
- Analice las soluciones propuestas

