

## Estructura general de un programa en C

```

/*    Comentarios de un parrafo completo
      comprendidos entre /*.....*/, sirven para
      aclarar qué el programa o una parte del programa */
//    Comentarios de 1 sola línea

//    Zona de ficheros de cabecera de las librerías

#include <..... ■ h>          // h de Head
#include <..... ■ h>

//    Zona de prototipos de funciones

int Potencia (int x,y)

//    Zona de variables globales

int valor;
float media_total;

void main (void)          // Prog. ppal. típico de Turbo C
{
    // llave de inicio del programa
    // codigo del programa
    .....
    .....
    .....
    // fin del programa
}

//    Desarrollo del código de las funciones anteriores

```

## **Modificadores de los tipos de datos básicos**

### **Datos tipo enteros:**

<b>unsigned:</b>	sin signo
<b>signed:</b>	con signo
<b>unsigned short:</b>	corto sin signo
<b>signed short:</b>	corto con signo
<b>unsigned long:</b>	largo sin signo
<b>signed long:</b>	largo con signo

### **Datos tipo carácter**

<b>unsigned:</b>	sin signo
<b>signed:</b>	con signo

### **Datos tipo real**

<b>double:</b>	doble
<b>signed:</b>	doble largo

## TIPOS ENTEROS

### **CHAR (CHARACTER)**

1 BYTE

RANGO: -128... 127 char  
0...255 unsigned char

EJEMPLO: **char** car; /\* car es una variable de tipo char \*/

### **INT (ENTERO)**

RANGO PARA 16 BITS: -32768... 32767 int  
0... 65535 unsigned int

EJEMPLO: **int** x; /\* x es de tipo entero \*/

## SHORT (ENTERO CORTO)

2 BYTES

RANGO: -32768... 32767 short  
0... 65535 unsigned short

EJEMPLO: **short** x,y; /\* declara x e y como enteros cortos \*/

## LONG (ENTERO LARGO)

4 BYTES

RANGO: -2147483648... 2147483647 long  
0... 4.294.967.295 unsigned long

EJEMPLO: **long** var; /\* var es de tipo long\*/

## ENUM (ENUMERACIÓN)

Tipo ENUMERADO

LISTA DE VALORES REPRESENTADOS POR IDENTIFICADORES

EJEMPLO :

**enum** semana

{ lunes, martes, miercoles, jueves, viernes, sabado, domingo} ;

enum semana ayer /\* ayer es un tipo enumerado semana \*/  
/\* lunes toma el valor 0 y domingo el valor 6\* /

## TIPO REAL

### **FLOAT** (REALES EN SIMPLE PRECISION)

4 BYTES

RANGO: -3.402823E+38.....-1.40129E45      negativos  
1.401293E-45 ... 3.40282E38      positivos

-3E+38.....-1'4E45  
1'4E-45.....3E38

EJEMPLO: **float** x;      /\* x es un real \*/

### **DOUBLE** (REALES EN DOBLE PRECISION)

8 BYTES

RANGO: -1.79769313316E308.....4.94065E-324      negativos  
4.94065E-324... 1.797334862316E308      positivos

EJEMPLO: **double** x;      /\* x es un real en doble precisión \*/

## ARRAYS

CONJUNTO DE ELEMENTOS DEL MISMO TIPO

EJEMPLO: `char a[40]; /* 40 caracteres del 0 al 39 */`

## VOID

SE UTILIZA PARA DECLARAR FUNCIONES QUE NO  
RETORNAN NINGUN VALOR O NO ACEPTAN PARAMETROS .

EJEMPLO: `void f (int a); /* la función no retorna valores */`

## CONST

INDICA QUE EL VALOR DE UN IDENTIFICADOR NO PUEDE  
SER MODIFICADO

## OPERADORES 'ESPECIALES'

**X ++** INCREMENTO DE LA VARIABLE X EN 1

**X --** DECREMENTO DE LA VARIABLE X EN 1

## ENTRADA - SALIDA ESTANDAR

Con este epígrafe nos referimos a las funciones estándar de C para realizar entrada de datos por teclado y salida de datos hacia pantalla.

Son funciones definidas en la librería estándar. Para usarlas es necesario incluir el fichero de cabecera de las funciones :

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

Estas funciones son:

<b>printf( )</b>	Salida de datos con formato
<b>scanf( )</b>	Entrada de datos con formato
<b>getchar( )</b>	Entrada de caracteres. 1 caracter
<b>putchar( )</b>	Salida de caracteres. 1 caracter
<b>fflush( )</b>	Borrado del buffer del teclado.

funcion **printf ( )**

Escribe una serie de caracteres en la salida estándar (pantalla).

Prototipo

```
int printf (const char *formato [,argumentoJ...]);
```

Devuelve

Número de caracteres escritos.

## Parámetros

**Formato:** Cadena de caracteres, entre comillas dobles, que especifica cómo va a ser la salida. Incluye caracteres y especificaciones de formato.

**Argumento:** Indica las variables a escribir.

Ejemplo:

```
float pi=3.141596;           // declara y asigna a la vez
printf("El número pi vale %f ",pi);
```

^ carácter de control %f

## Especificaciones de formato:

`% [ancho] [ . precision]`

**ancho:** Mínimo número de posiciones reservadas para la salida.

**precisión:** Mínimo número de posiciones reservadas para la parte decimal.

## Caracteres de control en función del tipo

Cuando necesitamos especificar el valor de una variable dentro de una instrucción **printf** debemos poner un **carácter de control** que indica **qué** tipo de dato va en esa posición:

control	Tipo asociado
<b>%d</b>	<b>int</b> enteros con signo base 10
<b>%u</b>	<b>unsigned int</b> enteros sin signo base 10
<b>%x</b>	<b>int</b> base 16
<b>%f</b>	<b>float</b> formato eee.ddd
<b>%e</b>	<b>float</b> formato xxxEee
<b>%c</b>	<b>char</b> caracteres
<b>%s</b>	<b>string</b> cadena de caracteres
<b>\n</b>	salto de línea
<b>\t</b>	tabulador a la derecha
<b>\a</b>	beep !
<b>h</b>	short
<b>l</b>	long → Modificadores
<b>L</b>	double

### funcion **scanf ( )**

Lee datos de la entrada estándar (teclado) , los interpreta y los almacena en los argumentos. Incluye caracteres y especificaciones de formato.

## Prototipo

int **scanf** (conts char *\*formato* [,*argumento*]...);

## Devuelve:

El número de datos leídos. Si es cero indica que no han sido asignados datos.

## Parámetros

**Formato:** Cadena de caracteres, entre comillas dobles, que especifica cómo van a ser introducidos los datos. Según se especifique en el formato , **así deberán ser introducidos los datos.**

**Argumento:** Indica las direcciones de variables que almacenarán los datos (&).

```
int a; float b; char c;
```

Sentencia	Entrada de datos
<code>scanf("%d %f %c" , &amp;a, &amp;b , &amp;c) ;</code>	5 2.3 b
<code>scanf("%d, %f , %c",&amp;a,&amp;b,&amp;c);</code>	5 , 23.4 , b
<code>scanf("%d : %f : %c",&amp;a,&amp;b,&amp;c);</code>	5 : 23.4 : b

&a → Dirección de memoria de la variable a

Ejemplo:

```
#include <stdio.h>

void main (void)
{
    int a,r;                // declaramos dos enteros
    float b;                //declaramos un real
    char c, s[20];          // declaramos un carácter y una cadena

    printf ("Introducir un entero, un real y un carácter : \n");
    r=scanf ("%d %f %c", &a, &b, &c);
    printf ("Nº de datos leidos: %d",r);
    printf ("Datos leidos: %d %f %c",a,b,c);
}
```

### Lectura de cadenas

```
#include <stdio.h>

void main(void)
{
    char nombre[20], apellido[20];
    printf ("Introduce el nombre: ");
    scanf ("%s", nombre);          //solo lee hasta el primer espacio en blanco
                                    // en una matriz ya pasamos la direccion

    fflush (stdin);
    printf ("\nApellidos: ");
    scanf ( " % [ ^ \ n] " ,apellido);
    printf ("\n\nNombre y Apellidos: %s %s ",nombre,apellido);
}
```

Si metemos en nombre: Juan Jose y en apellidos: Lopez Perez aparecera escrito Juan Lopez Perez

## funcion **getchar ( )**

Lee un caracter de la entrada estandar .

Prototipo:

```
int getchar(void);
```

Devuelve:

El caracter leido o un EOF si se detecta el fin de fichero.

## funcion **putchar ( )**

Escribe un caracter en la salida estandar .

Prototipo

```
int putchar(int c );
```

//recordar que un carácter es como un  
entero de 0..255

Devuelve

El caracter escrito o EOF si ocurre un error.

Parametros

c      caracter a imprimir.

## funcion **fflush (stdin)**

Borra el buffer del teclado.



## Estructuras de control

Selección incompleta **if <condición> acción**

```
if ( condición ) {  
    ...,  
    ...,  
}
```

```
if ( condición ) sentencia;
```

```
if ( pepe > 0 ) printf("\n PEPE es positivo ");
```

```
if ( pepe != 0 ) { pepe = 0;  
    printf("\n El valor es %f", pepe);  
}
```

```
if ( pepe ) { ...  
}
```



```

#include < stdio.h >
#define SI      1           //similar a una etiqueta con un valor
#define NO     0
void main(void)
{   char ch;
    int mayu, numero, otro;
    mayu=numero=otro= NO   // las igualamos todas a NO
    printf("\n Introduzca un caracter...");
    scanf(" %c", &ch);
    if ( ch >= 'A'    &&  ch <= 'Z' ) mayu = SI
        else if ( ch >= '0' && ch <= '9') numero = SI;
        else  otro = SI

}

```

---

```

#include < stdio.h >
void main(void) /* Menor de 3 numeros * /
{   int x,y ,z;
    printf( " \nIntroduzca los 3 numeros. .." ) ;
    scanf(%d %d %d", &x, &y, &z);
    if (x <y)
        if (x <z)   printf("\n El menor es %d", x);
        else       printf("\n El menor es %d", z);
    else if (y <z)  printf("\n El menores %d", y);
        else       printf("\n El menor es %d" , z);

}

```

## Selección incompleta

**Operador condicional** `<cond> ? accion1: accion2`

Es equivalente a

```
if (cond) accion1
    else accion2
```

```
maximo = (a > b) ? a: b;
if ( a > b ) maximo = a;
    else maximo = b;
```

```
absoluto = ( conta > 0 ) ? conta : -conta;
printf ("\n El mayor es %d ", (a>b)? a:b); )
```

```
#include < stdio.h >
#define SI      1
#define NO     0
void main(void)
{
    int anno, bisiesto;
    printf("\n Introduzca el año ");
    scanf(" %d" , anno);
    bisiesto=(anno%4== 0 && anno% 100 != 100 || anno %400== =0) ? SI:NO;
    if (bisiesto)      printf ("\n SI");
        else printf ("\n NO ");
}
```

## Estructura de Selección Múltiple

**switch (opcion) ...**

```

switch ( opcion)
{
    case 1:    ...;
              break;
    case 2:    ...;
              break;
    case 5:    ...;
              break;
    default:   ....;
}

```

Ejemplo:

```

#include < stdio.h >
void main (void)
{
    int numero ;
    printf("\n Introduzca un numero ");
    scanf('1 %d', &numero);
    switch(numero )
    {
        case 0:    printf("\n Es un CERO ");
                  break;
        case 1:    printf("\n Es un UNO ");
                  break;
        default:   printf("\n No ni CERO ni UNO ");
    }
}

```

```

#include < stdio.h >
void main(void)
{
    float salida[100] ,t;
    int opcion;
    printf("\n Introduzca una opcion ");
    scanf(" %d" , &opcion);
    switch( opcion )
        {case 0: /* Rampa */
            for(t=0; t<100;t++) salida[t]=t;
            break;
        case 1: /* Triangular */
            for ( t=0; t < 50; t+ +) salida[t] =t;
            for(t=50; t<100; t++) salida[t]= 100-t; break;
        case 2: /* Polarizada */
            for ( t=0; t < 50; t+ +) salida[t] =t;
            for ( t=50; t < 100; t+ + ) salida[t] = t-50;
            break;
        default: printf("\n NINGUNO DE ELLAS ");
        }
    for( t=0; t< 100; t+ +) printf("\n Valor = %f", salida[t]);
}

```

¿Cómo sería una cuadrada normal y otra polarizada?

Estructura repetitiva

**while (condicion) accion1;**

Bucle con condición de entrada.

ESTRUCTURA:

while ( condicion )

```

    {   .....
        .....
    }

```

while ( condicion ) accion;

Ejemplos:

```

#include <stdio.h>
void main (void)
{   int conta = 0;
    while ( conta < 10 )
        {   printf ("\n El valor es %d", conta );
            conta = conta + 1 ;
        }
    printf("\n Ha terminado el bucle. Conta = %d", conta );
}

```

```

#include <stdio.h>
void main(void)
{   int conta = 0;
    while ( conta < 10)           /* Bucle infinito */
        {   printf ("\n El valor es %d", conta );
            conta = conta -1;    // conta--
        }
}

```

Calculo del cuadrado de un numero:

```
#include <stdio. h>
void main(void)
{
    int numero = 0;
    while ( numero < 10)
        {
            printf ("\n El cuadrado es %d", numero*numero );
            numero++;
        }
}
```

Otra versión:

```
#include <stdio.h>
void main(void)
{
    int numero = 0;
    while ( numero*numero < 100 )
        {
            printf ("\n El cuadrado es %d", numero*numero );
            numero++;
        }
}
```

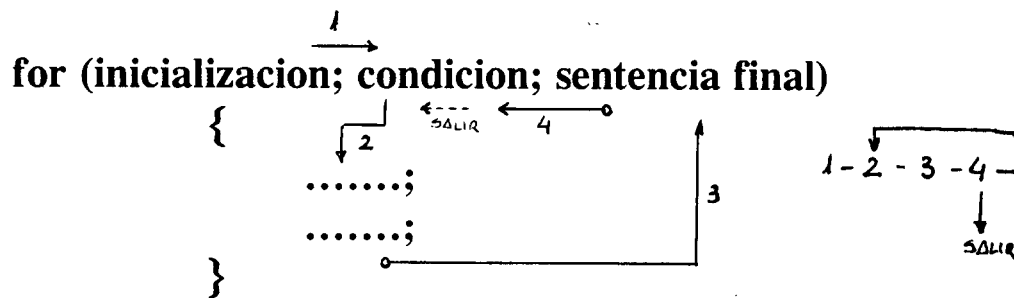
Lectura teclado:

```
#include <stdio. h>
void main(void)
{
    int numero, estado;
    estado = scanf("%d", &numero);
    while (estado==1)
        {
            printf("\n Numero %d", numero);
            estado = scanf("%d", &numero);
        }
}
```

```
#include <stdio.h>
void main(void)
{
    int numero;
    while ( scanf("%d", &numero) ==1)
        printf ("\n Numero %d", numero);
}
```

```
#include <stdio.h >
void main(void)
{
    int a;
    a=3;
    if (a) {        printf ("\n Valor de a %d." ,a );
                  a=0;
                }
    if (a)        printf ("\n Valor de a = %d.",a );
}
```

Estructura repetitiva **for ( i=... , i=... , incr i ) accion1;**



Ejemplos:

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

```
void main(void)
```

```
{ int conta;
```

```
for( conta = 0; conta < 10; conta = conta +1 )
```

```
printf ("\n El valor es %d", conta ); J
```

```
printf("\n Ha terminado el bucle. Conta = %d", conta );
```

```
}
```

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

```
void main(void)
```

```
{ int conta; /* No entra al bucle */
```

```
for ( conta = 10; conta < 10; conta++)
```

```
{ printf ("\n El valor es %d", conta );
```

```
conta = conta + 1 ;
```

```
}
```

```
printf ("\n Salida del bucle: %d", conta );
```

```
}
```

```
#include <stdio.h>
void main(void)
{
    int conta=0;
    for (; conta < 10; conta++) // inicializo fuera del bucle
        printf ("\n El valor es %d", conta );

    printf ("\n Salida del bucle: %d", conta );
}
```

Estructura repetitiva **do accion while (condicon);**

Bucle con condición de salida.

ESTRUCTURA:

```
do {
    .....
} while ( condicion ) ;
```

Ejemplos:

```
#include <stdio. h>
void main(void)
{
    int conta=0;
    do {    conta++;
        printf ("\n El valor es %d", conta );
    } while ( conta < 10);
}
```

```
#include <stdio.h>
void main(void)
{
    char ch;
    do { scanf("%c", &ch);
        printf("\n%c",ch);
    } while ( ch != ' * ');
}
```

```
#include <stdio.h >
void main(void)
{
    int t,k;
    for ( t=0; t<10; t++)
        { printf("\n El valor de t es %d'., t);
          for( k = 3; k >0; k--)
              printf("\n El valor de t+k es %d'., t+k);
        }
}
```

```
#include <stdio.h >
void main(void) /* Factoriales de los numeros entre 1 y 5
{
    int t, aux;
    int fac;
    for (t=1; t<=5; t++)
        {
            aux = t;
            fac = 1;
            do { fac = fac * aux.
                aux--; // Rutina de factorial
            } while ( aux > 0);
            printf('\n E1 factorial de %d es %d", t, fac);
        }
}
```

```
Salida: 1          1.2
        1.3.2      1.4.3.2
        1.5.4.3.2
```

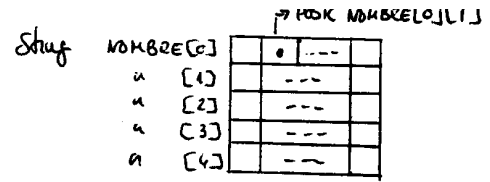
## Arrays ( Matrices y Vectores )

Variables del mismo tipo, con el mismo nombre y que se Distinguen y referencian por un indice :

```
float numero[10] ;           // Matriz de 10 n° reales
int k[104];                 // Matriz de 104 enteros
char ch[300] ;             // Cadena de 299 caracteres
int M [23][43]             // Matriz de 2 dimensiones
```

```
#include < stdio.h >
void main (void)
{   float k[5];
    int t;
        for ( t=0; t <5; t+ +)
        {   printf ("\n Introduzca valor de k[%d] ",t);
            scanf(" %f",&k[ t ]);
        }
    for( t=4; t > =0; t--)
        printf ("\n Los valores han sido %f ", k[ t ]);
}
```

```
#include < stdio.h >
void main (void)
{
    char nombre[5][45];
    int t;
    for ( t=0; t<5; t+ + )
        {
            printf("\n Introduzca nombre de k[%d] ", t);
            scanf("%s", &nombre[t]);
        }
    for( t=4; t> =0; t-- )
        printf("\n Los nombres han sido %s", nombre[t]);
}
```



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

```
void main(void)
```

```
{
    int k[3][3];          // Matriz de dimensiones 3x3
    int x,y;
    printf ("\n Introduzca los valores de la matriz...");
    for ( x=0; x < 3; x+ + )
        for ( y=0; y <3;y+ + )
            { printf ( "\n Coeficiente [%d][%d] " , x,y);
              scanf ("%d", &k[x][y]);
            }

    printf ("\n La matriz es: \n");

    // Ahora escribimos el contenido de la matriz ...
```

```

for (x=0; x<3; x+ + )
    {
        for (y=0; y<3;y++)
            printf(" %d ", k[x][y]);
            printf("\n");
        }
printf(tl\n La matriz traspuesta es: \ntl);
for (x=0; x<3; x+ + )
    {
        for( y=0; y <3;y+ + )
            printf("%d", k[y][x]);
            printf("\n");
        }
    }
}

```

## Strings o cadenas de caracteres

Es una array de dos dimensiones de tipo char donde cada fila se considera una cadena de caracteres.

H	O	L	A	
C	O	M	O	L
V	O	Y		
C	A	S	A	

List [filas][columnas]

Lista[0]

Lista[1]

Lista[2]

Lista[3]

Ejemplo: Leer una lista de nombres y almacenarlos en un array

```
#include<stdio.h>

void main(void)
#define max 10          //      Se pone así porque es mas facil, si queremos
#define long 60        //      posteriormente modificar las dimensiones de la matriz

void main(void)
{   char lista [max][long];
    int i=0,n;
    puts ("Pulsa ^Z para finalizar...");
    do
        {   printf ("\nNombre y apellidos...");
            fin=gets(Lista[i]);
            i++;
        }while (fin!= NULL && i<max);

    printf ("\n\n");

    for (n=0;n<i;n++)
        printf ("\n Nombre y Apellidos..: %s",Lista[n]);
}

//      Recordar lo que ocurría con scanf al leer una cadena con espacios en blanco
```

# FUNCIONES

Subrutinas o subprogramas que forman un programa.

## FUNCIONES SIN ARGUMENTOS

Son funciones que no recogen ni devuelven ningún valor.

```
#include < stdio.h >
void func1(void);           /* Prototipo*/

    ^----- no recoge ningun parametro
    |
    |----- no devuelve ningun valor

void main (void)
{   int a,b, conta;
    a=1; b=2;
    func1 ( );             // LLamada a la función 1
    for (conta = 0; conta < 10; conta+ + )
        func1 ( );
}

//   Cuerpo o codigo de la funcion

void func1( )              // Definición
{   printf("\n Esta funcion imprime un mensaje ");
}
}
```

## FUNCIONES CON ARGUMENTOS

Son funciones que reciben 1 o más parámetros del mismo o distinto tipo.

```
#include <stdio.h >

void func1(int, int);          /* Prototipo*/

                                ^----- recoge dos parámetros de tipo entero

void main ( )
{
    int a,b, conta;
    a=1; b=2;
    func1(a,b);               //Llamada a la función
    for(conta = 0; conta < 10; conta + + )
        func1(a,conta);
}

void func1(int x, int y)
{
    printf ("\nValores : %d, %d", x, y ");
}
```

## FUNCION QUE DEVUELVE UN VALOR

```
#include < stdio.h >
double suma(double, double);
int maximo(int,int);
void main (void)
{
    int a,b, mayor;
    double x, y ,total;
    x=22.2; y= 443.0;
    a=1; b=2;
    total = suma (x,y);
    mayor= maximo (a,b);
}

double suma(double x, double y)
{
    return (x+y);
}

int maximo(int x, int y)
{
    // si (x>y) entonces devuelve x, sino devuelve y
    return (x >y) ? x:y;
}
```

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

void uno( )
{
    printf("\n UNO 1");
    dos( );           // desde una funcion llamamos a otra funcion
}

void dos( )
{
    printf("\n DOS 2");
}
```

## Variables locales y globales

### Locales

```
#include < stdio.h>
void fun(void);
void main (void )
{
    int a,b;                /* Variables locales */
    a=b= 1;
    printf("\n Antes de llamar a fun: a = %d, b= %d", a,b);
    fun( );
    printf("\n Despues de llamar a fun: a = %d, b= %d",a,b);
}

void fun()
{
    int a,b;                // son locales a esta funcion
    a=b=11;
    printf {"\n En la función: a = %d  b= %d", a,b);
}
```

---

```
#include < stdio.h >
int suma(int,int);
void main(void)
{
    int a,b,total;        /* Variables locales */
    a=b=1;
    total = suma(a,b);
    printf ("\n a = %d, b= %d, suma = %d",a,b,total);
}

int suma(int a, int b)
{
    int total;            //a y b son locales en suma
    total = a+b;
    a++;
    b++;
    return ( total) ;
}
```

## Globales

```
#include < stdio.h >
void suma(int,int);
int total;          /* Variable global , declarada fuera del main */
void main (void)
{
    int a,b;      /* Variables locales */
    a=b=1;
    suma(a,b);
    printf("\n a = %d, b= %d, suma = %d1",a,b,total);
}
void suma(int a, int b )
{
    /* a y b son locales en suma */
    total = a+b;    /* Total es variable global */
    a++; b++;
}
}
```

## Punteros

Una variable puede ser accedida de dos formas distintas.

Recordemos que una variable no es mas que una posición de memoria que se reserva para almacenar un valor, y que se asigna en el momento de la declaración

Ejemplo:

int x=3;                      direccion RAM 12300    →    3 X   ← Identificador

Esto significa que la variable `X` puede ser accedida por su identificador o por la dirección RAM que ocupa 12300.

Como la dirección se asigna cuando se ejecuta un programa y puede variar de una ejecución a otra, necesitamos una **variable** que pueda contener a una dirección y un **operador** que nos diga que dirección RAM tiene asignada una variable. Esto lo realiza un **PUNTERO**. Los operadores son `*` y `&`.

- `&` aplicado a una variable obtiene la dirección RAM
- `*` aplicado a un puntero obtiene el contenido de la dirección de la variable a la que apunta.

### Declaración:

```
int *p;           // Declara una variable de tipo dirección
int x=3;
p= &x;           // Obtenemos la dirección de la variable X
*p=4             // El contenido de lo que apunta p vale 4
                // Es lo mismo que si hubiera hecho x=4
```

Los punteros nos sirven, entre otras cosas para que una función pueda cambiar los valores de una variable local y también para que pueda devolver más de un argumento

Ejemplo:

```
#include "stdio.h"
void cambia (int *, int*);
void main ( void)
{   int x = 1;
    int y= 2;
    cambia(&x,&y) // pasamos las direcciones de memoria de x e y
    printf("\n El valor de x = %d y de y = %d ", x,y);
}
```

```
void cambia ( int *x, int *y )
{   int aux;
    aux= *x;           //GUARDO EN AUX EL CONTENIDO DE DONDE APUNTA X
    *x = *y
    *y = aux          ← Hacer dibujo en pizarra
}
```

x

y

aux

→ funcion

a

b

→ main

## PUNTEROS y ARRAYS.

La relacion entre un puntero y un array es que la variable que designa al array contiene la direccion de memoria del primer elemento del mismo.

```
#include "stdio.h"
void main (void)
{
    int x[10] , t;
    for ( t=0; t<10; t++)
        x[t] = t;
    for ( t=0; t<10; t++)
        printf("\n Valores : %d,  %d' ", x[t], *(x+t));
}
```

```
#include "stdio.h"
void main (void)
{
    int x[10], t;
    int *p;
    p = &x[0];    //Obtengo la dir del primer elem.
    for ( t=0; t<10; t++)
        x[t] = t;
    for ( t=0; t<10; t++)
        {
            printf("\n El valor es %d y %d", x[t], *p);
            p++;
        }
}
```

## Devolución de más de 1 valor

```

#include <stdio.h>

void F (   int,   int,   int*,   int* )
          ↑   ↑   ↑↓   ↓

void main (void)
{
    int x,y,z,t;
    x=1; y=5; z=10;
    F ( x, y, &z, &t );
    printf ("%d - %d - %d - %d ",x, y, z, t);
}

          x   y   &z   &t
void F (   int a, int b, int *c,int *d)
          1   5   10   ?

{
    int aux,m,n;
    aux=a+b;
    m= b * (*c)+aux;
    n= (*c)+m;
    *c=m;
    *d=n;
}

```

## Cadenas de caracteres

Es un vector de caracteres. Las cadenas acaban con el carácter de fin de cadena '\0', el cual no es visible ni imprimible.

Ejemplos:

```
char cadena [ ] ="abcde";           // es un vector de 6 posiciones, la ultima es '\0'
char cadena [3]="pepe";             // error pues supera los límites
char nombre [30]="pepe"             // el resto se inicializa a '\0'
char nombres [100][60];            // Define una matriz de 100 cadenas y 59
                                    caracteres de longitud
```

Funcion **gets ( )**

Lee una cadena desde el teclado, Sustituye el carácter del [enter] por el '\0' de fin de cadena. Lee espacios en blanco.

prototipo:

```
char * gets (char * cadena)
```

Parametros:

cadena: Vector donde se almacenara la cadena leída

Devuelve:

puntero a la cadena leída

## Función **puts ( )**

Escribe una cadena de caracteres en la pantalla, Sustituye el '\0' por un salto de línea (\n).

Prototipo:

```
int puts (char* cadena)
```

Parámetros:

cadena: cadena a presentar por pantalla

Devuelve : un valor positivo si no hay error. Si hay error devuelve EOF.

Ejemplo:

```
#include <stdio.h>
#include <conio.h>
char linea[81], *pc;
void main (void)
{
    printf ("Introducir una cadena");
    pc=gets(linea);
    printf ("\n Pulsa una tecla para continuar");
    getch ( );
    printf (" Presento la cadena");
    puts (pc);
    puts (linea);
}
```

## Funciones para el manejo de cadenas

Librería <string.h>

Función	Observaciones
strcat(cad1,cad2)	Añade la cad2 a la cad1. Termina con un '\0' y devuelve un puntero a cad1.
<del>NO</del> strchr(cad,c)	Devuelve un puntero a la primera aparición de c en cad.
strcmp(cad1,cad2)	Compara cad1 y cad2. Devuelve <0 si cad1>cad2; =0 si cad1=cad2; >0 si cad1<cad2.
strcpy(cad1,cad2)	Copia cad2 en cad1. Devuelve un puntero a cad1.
strlen(cad)	Devuelve la longitud en bytes de cad.
strncat(cad1,cad2,n)	Añade los primeros n caracteres de cad2 a cad1.
strncmp(cad1,cad2,n)	Compara los n primeros caracteres de cad1 y cad2. Devuelve <0 si cad1>cad2; =0 si cad1=cad2; >0 si cad1<cad2.
strncpy(cad1,cad2,n)	Copia los n primeros caracteres de cad2 en cad1. Devuelve un puntero a cad1.
<del>NO</del> strrchr(cad,n)	Devuelve un puntero a la última aparición de c en cad.
strlwr(cad)	Convierte la cad a minúsculas.
strupr(cad)	Convierte la cad a mayúsculas.

```
#include <stdio.h >
#include <string.h >
void main(void)
{
    char origen[ ] = " SSSS323232 *S " ;
    char destino[100];
    printf( " \n % s " , origen ) ;
    strcpy(destino,origen);           // copia dos cadenas
    printf("\n %s" ,destino);
    printf("\n La longitud del array origen es %d" , strlen(origen));
    if (strcmp(origen,destino))      // compara dos cadenas
        printf("\n Son diferentes ");
    else printf("\n Son iguales");
    destino[2] = '@' ;
    if (strcmp(origen,destino) printf("\n Son diferentes ");
        else printf( " \n Son iguales " ) ;
    strcat(destino,origen);         /* Concatena dos cadenas */
    printf("\n %s" ,destino);
    strncat(destino,origen,3);      /*Concatena los 3 primeros de origen */
    printf("\n% s " ,destino );
    strset(destino, '\0');         /*Setea la cadena a un caracter dado */
    printf("\n %s" ,destino);
}
```

## Funciones de conversión entre números y caracteres

Función	Observaciones
atof(cad)	Convierte una cadena a un real double.
atoi(cad)	Convierte una cadena a un entero int.
atol(cad)	Convierte una cadena a un entero long.
fcvt(valor, decs, pdec, signo)	Convierte un real double a una cadena <i>10001 NO PONE EL PUNTO DECIMAL : 3.1415 → 31415</i>
itoa(valor, cadena, base)	Convierte un entero int a una cadena <i>→ base 10</i>
ltoa(valor, cadena, base)	Convierte un entero long a una cadena

## Funciones avanzadas para la presentación de texto por pantalla

- int **cprintf** (char \*formato [, argumentos, ...]);
- int **cputs** (char \*cadena);
  
- (void) **normvideo** (void)
- (void) **highvideo** (void)
- (void) **lowvideo** (void)
  
- (void) **textcolor** (int color)
- (void) **textbackground** (int color)
  
- (void) **gotoxy** (int x , int y)
- (void) **puttext** (int izda, int arriba, int dcha, int abajo, void\* dato)

## APENDICE RESUMEN DE LAS FUNCIONES PRINCIPALES DE LAS UNIDADES ESTANDAR DE C

- FUNCIONES CON CADENAS.
- FUNCIONES MATEMATICAS.
- FUNCIONES DE HORA Y FECHA Y OTRAS RELACIONADAS CON EL SISTEMA.
- FUNCIONES DE PANTALLA
- FUNCIONES DE ENTRADA Y DE SALIDA
- OTRAS FUNCIONES

### FUNCIONES CON CADENAS

```
include <stdlib.h>
    double atof(char *cad)
```

#### Descripción

Esta función devuelve la cadena apuntada por cad a un valor de tipo double. La cadena debe tener un número válido en coma flotante. Esto incluye espacios en blanco, signos de puntuación distintos el punto, y caracteres que no sean E o e. Esto supone que si atof() se llama con la cadena "100.00HOLA" se devuelve el valor 100.00.

```
#include<stdlib.h>
    int atoi(char *cad)
```

#### Descripción

La función atoi() convierte la cadena apuntada por cad a un valor int. La cadena debe contener un número entero válido. Si este no es el caso, el valor devuelto queda indefinido; sin embargo, muchas de las implementaciones devuelven cero.

El número puede acabar con cualquier caracter que no forme parte de un número entero. Esto incluye espacios en blanco, signos de puntuación y otros que no sean dígitos. Esto supone que si atoi() se llama con 123.23 se devuelve el valor entero 123 y el .23 se ignora.

```
#include<stdlib.h>
int atol(char *cad)
```

#### Descripción

La función `atol()` convierte la cadena apuntada por `cad` a un valor `long int`. La cadena debe contener un número entero de tipo `long` válido. Si no es este el caso, el valor devuelto queda indefinido; sin embargo, la mayor parte de las implementaciones devuelven cero.

El número puede acabar con cualquier carácter que no forme parte de un número entero. Esto incluye espacios en blanco, signos de puntuación y otros que no sean dígitos. Esto supone que si `atol()` se llama con `123.23` se devuelve el valor entero `123` y el `.23` se ignora.

```
#include<stdlib.h>
int itoa( int num, char cad, int base)
```

#### Descripción

La función `itoa()` convierte el entero `num` a su cadena equivalente y sitúa el resultado en la cadena apuntada por `cad`. La base de la cadena de salida se determina por `base`, que se encuentra normalmente en el rango 2 a 16. La función `itoa` devuelve un puntero a `cad`. Generalmente no devuelve valor de error. Asegúrese al llamar a `itoa()` que la cadena es lo suficientemente grande como para contener el resultado transformado. Su uso principal es convertir tipos de datos enteros a cadenas de modo que pueden ser enviados a un dispositivo no soportado directamente por el sistema de E/S usual de C -es decir, a un dispositivo no de flujo-. Lo mismo se puede utilizar llevando a cabo utilizando `sprintf()`.

```
#include<stdlib.h>
int ltoa( int long num, char cad, int base)
```

#### Descripción

La función `ltoa()` convierte el entero `num` de tipo `long` a su cadena equivalente y sitúa el resultado en la cadena apuntada por `cad`. La base de la cadena de salida se determina por `base`, que se encuentra normalmente en el rango 2 a 16.

La función `ltoa()` devuelve un puntero a `cad`. Generalmente no devuelve valor de error. Asegúrese al llamar a `ltoa()` que la cadena es lo suficientemente grande como para contener el resultado transformado.

Su uso principal es transformar enteros en cadenas de modo que pueden ser enviados a un dispositivo no soportado directamente por el sistema de E/S usual de C -es decir, a un dispositivo de no flujo-. Lo mismo se puede utilizar llevando a cabo utilizando `sprintf()`.

```
#include<stdlib.h>
```

```
long int strtol(char *ini, char *fin, int base)
```

#### Descripción

Convierte la representación en cadena de caracteres de un número-almacenada en la cadena apuntada por `ini`- en un número de tipo `long int` y devuelve el resultado. La base del número está determinada por `base`. Si `base` es 0, debe estar en el rango de 2 a 36.

La función `strtol()` trabaja de la siguiente forma: primero elimina cualquier espacio en blanco de la cadena apuntada por `ini`. A continuación, se lee cada uno de los caracteres que constituyen el número. Cualquier caracter que no pueda formar parte de un número de tipo `long int` finaliza el proceso. Por último, `fin` se deja apuntando al resto, si lo hay, de la cadena original. Esto supone que si `strol()` se llama con 100.000 pliers, se devuelve el valor de 100L y `fin` apunta al espacio que precede a pliers.

```
#include <ctype.h>
```

```
int isalnum(int ch)
```

#### Descripción

`isalnum()` es una función que analiza a `ch` y devuelve un valor distinto de cero en caso de que ésta sea una letra del alfabeto o un dígito; si no, devuelve cero.

```
#include<ctype.h>
```

```
int isalpha(int ch)
```

#### Descripción

`isalpha()` es una función que nos sirve para determinar si `ch` es una letra del alfabeto; si es así, regresa un valor diferente de cero como resultado de la ejecución, en cualquier otro caso, devuelve cero.

```
#include <ctype.h>
int iscntrl(int ch)
```

#### Descripción

Mediante esta función, determinamos si `ch` es un carácter de control entre 0 y 0x1F (0 a 31 en decimal) o si `ch` es igual a 0x7F (la tecla DEL) en cualquier todo caso, devuelve cero.

```
#include <ctype.h>
int isdigit(int ch)
```

#### Descripción

La función `isdigit()` devuelve un valor distinto de cero si `ch` es un número dígito (desde 0 a 9). De lo contrario, devuelve cero.

```
#include <ctype.h>
int islower(int ch)
```

#### Descripción

La función `islower()` devuelve un valor distinto de cero cuando `ch` es un carácter en minúscula es decir, una letra desde `a` hasta `z`; en cualquier otro caso devuelve cero.

```
#include <ctype.h>
int isprint(int ch)
```

#### Descripción

La función `isprint()` verifica que `ch` sea un carácter imprimible, incluyendo el espacio en blanco; en cualquier otro caso, se devuelve cero. Generalmente, los caracteres imprimibles oscilan entre 0x20 y 0x7E.

```
#include <ctype.h>
int isspace(int ch)
```

#### Descripción

Esta función, devuelve un valor diferente de cero en caso de que `ch` sea el espacio en blanco, el tabulador o un carácter de salto de línea; en cualquier otro caso, devuelve cero.

```
#include<ctype.h>
int isupper(int ch);
```

#### Descripción

isupper() comprueba que ch sea una letra mayúscula(cualquier letra entre A y Z); en caso contrario devuelve cero.

```
#include <ctype.h>
int isxdigit(int ch)
```

#### Descripción

La función isxdigit() devuelve un valor distinto de cero si ch es un dígito hexadecimal; en caso contrario, devuelve cero. Los dígitos se encuentran en uno de estos rangos: A hasta F, a hasta f, y de 0 a 9.

```
#include <string.h>
char *strcat(char *cad1, char *cad2)
```

#### Descripción

Esta función concatena una copia de cad2 en cad1 y añade al final de cad1 un caracter nulo. El caracter nulo determinación que originalmente tenía cad1 es sustituido por el primer caracter de cad2. La cadena cad2 no se toca en esta operación. La función devuelve cad1. Es tu responsabilidad que cad1 sea lo suficientemente grande como para mantener su contenido original y el de cad2.

```
#include<string.h>
char *strchr(char *cad, int ch)
```

#### Descripción

La función strchr() devuelve un puntero a la primera ocurrencia de ch en la cadena apuntada por cad. Si no se encuentra, devuelve un puntero nulo.

```
#include<string.h>
int strcmp(char *cad1, char *cad2)
```

#### Descripción

Por medio de esta función se comparan lexicográficamente dos cadenas que finalizan con el caracter nulo y devuelve un entero que se interpreta de la siguiente forma:

Valor	Interpretación
Menor que 0	cad1 es menor que cad2
0	cad1 es iguala cad2
Mayor que 0	cad1 es mayor que cad2

```
#include <string.h>
char *strcpy(char *cad1, char *cad2)
```

#### Descripción

strcpy() se utiliza para copiar el contenido de cad2 en cad1. La variable cad2 debe ser un puntero a una cadena que finalice con un caracter nulo; strcpy() devuelve un puntero a cad1.

Si cad1 y cad2 se solapan, el comportamiento de strcpy() es indefinido.

```
#include <string.h>
char *strerror(int num_error)
```

#### Descripción

Esta función convierte el número de error especificado por num\_error en una cadena de mensaje de error. Regresa un putero a la cadena. El mensaje a cada num\_error está definido en la implementación. Bajo ninguna circunstancia se debe modificar la cadena.

```
#include<string.h>
unsigned int strlen(char *cad)
```

#### Descripción

strlen() nos sirve para determinar el número de caracteres que contine una cadena que finaliza con el caracter nulo. El carácter nulo no se contabiliza.

```
#include<string.h>
char *strncat(char *cad1,char *cad2, unsigned int n)
```

#### Descripción

La función `strncat()` concatena `n` caracteres de la cadena apuntada por `cad2` en la cadena apuntada por `cad1` y pone al final de `cad1` el caracter nulo. El caracter nulo que tenia inicialmente `cad1` es sustituido por el primer caracter de `cad2`. La cadena `cad2` no se toca. `strncat()` devuelve `cad1`. No olvide que no se hacen comprobaciones de límites. Así que es responsabilidad del programador asegurarse que `cad1` sea lo suficientemente grande como para mantener su contenido original y el de `cad2`.

```
#include<string.h>
int strncmp(char *cad1,char *cad2, unsigned int n )
```

#### Descripción

Esta función compara lexicográficamente un máximo de `n` caracteres de las dos cadenas terminadas con un caracter nulo y devuelve un entero cuyo significado se presenta a continuación:

Valor	Interpretación
Menos que 0	<code>cad1</code> es menor que <code>cad2</code>
0	<code>cad1</code> es iguala <code>cad2</code>
Mayor que 0	<code>cad1</code> es mator que <code>cad2</code>

```
#include<string.h>
char *strncpy(char *cad1,char *cad2, unsigned int n)
```

#### Descripción

La función `strncpy()` copia `n` caracteres de la cadena apuntada por `cad2` en la cadena apuntada por `cad1` . El elemento `cad2` debe de tener un caracter final de cadena nulo. La función devuelve un puntero a `cad1`. Los caracteres nulos al final, indican el final de la cadena.

Si `cad1` y `cad2` se solapan, el comportamiento de `strncpy()` está indefinido. Si la cadena apuntada por `cad2` contiene menos caracteres que los indicados por `n`, se añaden caracteres nulos al final de `cad1` hasta que se copie `n` caracteres. En cambio, si la cadena apuntada por `cad2` es mayor que cuenta, la cadena resultante apuntada por `cad1` no tiene caracter nulo de terminación.

```
#include<stdio.h>
char *strstr(char *cad1,char *cad2)
```

Descripción

La función strstr() devuelve un puntero a la dirección de la primera ocurrencia en la cadena apuntada por cad1 de la cadena apuntada por cad2 excepto el caracter nulo de terminación de cad2. Si no la encuentra, devuelve un puntero nulo.

```
#include<ctype.h>
int tolower(int ch)
```

Descripción

Mediante esta función se regresa el equivalente en minúscula de ch si ch es una letra; en cualquier otro caso devuelve ch sin modificar.

```
#include <ctype.h>
int toupper(int ch)
```

Descripción

toupper() devuelve la mayúscula de ch si ch es una letra; en cualquier otro, devuelve ch sin modificar.

## FUNCIONES MATEMATICAS

La mayoría de las funciones matemáticas estan definidas con tipo de datos dobles pero podemos utilizar esas funciones con tipos de datos de menor rango, excepto caracteres.

```
include <stdlib.h>
int abs(int num)
```

Descripción

La función abs() devuelve el valor absoluto del enteronum.

```
#include<math.h>
double acos(double arg)
```

#### Descripción

La función `acos()` obtiene el arcocoseno de `arg`. El argumento de `acos()` debe estar en el rango de -1 a 1; en cualquier otro caso se produce un error de dominio.

```
#include<math.h>
double asin(double arg)
```

#### Descripción

Esta función devuelve el arcoseno de `arg`. El parámetro con el que se mande llamar a `asin()` debe estar en el rango de -1 a 1 para que no produzca un error de dominio.

```
#include <math.h>
double atan(double arg)
```

#### Descripción

Devuelve el arcotangente de `arg`.

```
#include<math.h>
double atan2(double y, double x)
```

#### Descripción

`atan2()` devuelve el arcotangente de `y/x`. Utiliza el signo de sus argumentos para obtener el cuadrante del valor devuelto.

```
#include <math.h>
double ceil(double num)
```

#### Descripción

Esta función da como resultado un valor `double` que representa el entero más pequeño que es mayor o igual a `num`. Por ejemplo, dado 1.02, `ceil()` devuelve 2.0; dado -1.02, devuelve -1.

```
#include <math.h>
double cos(double arg)
```

#### Descripción

Esta función devuelve el coseno de arg. El valor de arg debe darse en radianes.

```
#include<math.h>
double cosh(double arg)
```

#### Descripción

cosh() devuelve el coseno hiperbólico de arg. El valor de arg debe darse en radianes.

```
#include <math.h>
double exp(double arg)
```

#### Descripción

La función exp() devuelve un número **e** elevado a la potencia arg.

```
#include <math.h>
double fabs(double num)
```

#### Descripción

Devuelve el valor absoluto de num.

```
#include<math.h>
double floor(double num)
```

#### Descripción

La función floor() el mayor entero (representado en double) que no es mayor que num. Por ejemplo, dado 1.02, floor() devuelve 1.0; dado -1.02, floor() devuelve -2.0

```
#include <math.h>
double fmod(double x, double y)
```

#### Descripción

Esta función retorna el residuo de la división entera x/y.

```
#include <math.h>
double log(double num)
```

#### Descripción

La función log() devuelve el **logaritmo neperiano** de num. Se produce un error cuando num es negativo y error de rango si el argumento es cero.

```
#include <math.h>
double log10(double arg)
```

#### Descripción

Esta función, regresa el **logaritmo decimal** de num. Se produce un error de dominio si el argumento es negativo, y un error de rango en caso de ser cero.

```
#include <math.h>
double modf(double num, int i)
```

#### Descripción

Esta función descompone num en su parte entera y fraccionaria, y situa la parte entera en la variable apuntada por i.

```
#include <math.h>
double pow(double base, double exp)
```

#### Descripción

pow() retorna base elevada a exp. Se produce un error de dominio si base es cero y exp es menor o igual a cero. También ocurre si base es negativo y exp no es entero. Un desbordamiento produce un error de rango.

```
#include <math.h>
double sin (double arg)
```

Descripción

Devuelve el seno de arg. El valor de arg, debe darse en radianes.

```
#include <math.h>
double sinh(double arg)
```

Descripción

La función sinh() devuelve el seno hiperbólico de arg. El valor de arg debe darse en radianes.

```
#include <math.h>
double sqrt(double num)
```

Descripción

La función sqrt() devuelve la raíz cuadrada de num. Si se llama con un número negativo, se produce un error de dominio.

```
#include <math.h>
double tan(double arg)
```

Descripción

Esta función, devuelve la tangente de arg. El valor de arg debe darse en radianes.

```
#include <math.h>
double tanh(double arg)
```

Descripción

tanh() retorna la tangente hiperbólica de arg. El valor devuelto debe darse en radianes.

## FUNCIONES DE PANTALLA

```
#include<conio.h>
void clrcol()
```

### Descripción

clrcol() borra desde la posición del cursor hasta que se alcanza el final de la línea en el extremo derecho de la pantalla. Esta operación sólo es válida en modo texto.

```
#include<conio.h>
void clreos()
```

### Descripción

La función clreos() borra la pantalla desde la posición actual del curso hasta abajo. Sólo se aplica en pantallas en modotexto.

```
#include<conio.h>
void clrscr()
```

### Descripción

La función clrscr() borra toda la pantalla. En general, esta operación sólo se aplica cuando la pantalla está en modo texto.

```
include<conio.h>
void gotoxy(int x, int y)
```

### Descripción

Esta función, sitúa el cursor en la fila y columna definidas por x,y. Esta operación sólo se utiliza en pantallas de modo texto.

```
#include<conio.h>
void home()
```

### Descripción

Coloca el cursor en la posición 0,0 (el extremo superior izquierdo de la pantalla). No borra la pantalla. Sólo funciona en modo texto.

```
#include<conio.h>
int gettext(int izq,int top, int der, int centro, void *destino);
```

#### Descripción

Copia texto de la pantalla en modo texto a la memoria. Las coordenadas utilizadas son absolutas, no relativas a la ventana en la que se esté trabajando. La coordenada del extremo superior izquierdo es (1,1). Regresa un valor diferente de cero si la operación fué exitosa.

```
#include<conio.h>
void inline();
```

#### Descripción

Inserta una línea en blanco en una ventana de texto en la posición del cursor. Las líneas que se encuentran debajo de la posición del cursor se recorren una línea hacia abajo y se pierde la última línea.

```
#include<conio.h>
void lowvideo(void);
```

#### Descripción

Disminuye la intensidad de la luminosidad de los caracteres. Afecta a todos los caracteres que se imprimen posteriormente en pantalla dentro de esa ventana.

```
#include<conio.h>
void highvideo(void);
```

#### Descripción

Aumenta la intensidad de la luminosidad de los caracteres. Afecta a todos los caracteres que se imprimen posteriormente en pantalla dentro de esa ventana.

```
#include<conio.h>
void normvideo(void);
```

#### Descripción

Anula a highvideo() y/o normvideo(). Es decir, la luminosidad de los caracteres que se escriban a continuación, será normal dentro de esa ventana.

```
#include<conio.h>
void textcolor(int color);
```

#### Descripción

Selecciona un nuevo color de los caracteres que se escribirán a continuación.

```
#include<conio.h>
void texbackground(int color);
```

#### Descripción

Selecciona un nuevo color del fondo de la pantalla.

```
#include<conio.h>
void window(int iniciox,int inicioy, int finx, int finy);
```

#### Descripción

Crea una ventana activa en pantalla.

## FUNCIONES DE ENTRADA Y SALIDA

```
#include<stdio.h>
int getchar()
```

#### Descripción

Esta función lee un caracter desde el teclado hasta que se pulse <ENTER>. En caso de que se digite más de un caracter antes de pulsar <ENTER>, la variable sólo almacenará el primer caracter que se tecleó.

```
#include<stdio.h>
char *gets(char *cad)
```

#### Descripción

Esta función lee una cadena de caracteres desde el teclado y la coloca en la cadena cad. Se leen caracteres hasta que se recibe la pulsación de <ENTER>. Esto no quiere decir que en la cadena se almacenará este caracter sino que añadirá un caracternulo (\0) para identificar el

final de la cadena. En caso de error, `gets()` retorna un puntero nulo y el contenido de `cad` será indeterminado.

Con `gets()` puedes leer todos los caracteres que desees. Por tanto, corresponde al programador asegurarse que la cantidad de caracteres leídos no superen la capacidad del array.

```
#include<stdio.h>
int putchar(ch)
```

#### Descripción

`putchar()` escribe un caracter en pantalla donde `ch` puede ser una variable de tipo caracter o un caracter ASCII entre comillas simples('a') y posiciona el cursor en la siguiente línea.

```
#include<stdio.h>
int puts( char *cad)
```

#### Descripción

La función `puts()` escribe en pantalla el contenido de una cadena apuntada por `cad` (un conjunto de caracteres) y posiciona el cursor en la siguiente línea.

```
#include<stdio.h>
int kbhit()
```

#### Descripción

Esta función no está definida por el ANSI propuesto. Sin embargo, la incluimos aquí porque aunque con nombre diferente, se encuentra definida en todas las implementaciones de C.

Su uso principal es permitir que el usuario pueda interrumpir alguna rutina desde el teclado. Regresa un valor distinto de cero, si se ha pulsado una tecla y en caso contrario, el valor retornado es cero.

```
#include<stdio.h>
int printf(char *formato, lista_arg)
```

### Descripción

La función printf() despliega en pantalla tanto cadenas constantes (mensajes) como variables de acuerdo al contenido de formato. Con formato, se especifica la cadena constante y/o el tipo de variables que desplegará en pantalla.

Todo esto siempre vá entre comillas dobles(" "). Por ejemplo:

```
printf("Hola, como estas?"); printf("%s",nombre);
```

Con la primera instrucción, mandamos un mensaje a pantalla. En este caso, no es necesario especificar algún formato ya que sólo se despliega el mensaje. Es en el segundo ejemplo de printf() donde utilizamos formato para determinar el tipo de variable(s) que van a ser desplegadas. En el caso anterior, determinamos que se va a escribir el contenido de nombre que es una variable de tipo cadena.

Veamos el formato que puede utilizarse para definir el contenido de cualquier variable:

Código	Formato
%c	Un sólo caracter
%d	Decimal
%i	Decimal
%e	Notación científica 5e-3
%f	Coma flotante
%g	utiliza el más corto de %e o %f.
%o	Octal
%s	Cadena de caracteres
%u	Decimal sin signo
%x	Hexadecimal
%%	Imprimir el símbolo %
%p	Presentar un puntero

También podemos desplegar en el monitor mensajes como el contenido de variables con el mismo comando printf() como en el ejemplo siguiente:

```
printf("Hola %s soy %s, tengo %i años."huesped,nombre,edad);
```

Si huesped="visitante", nombre="el supervisor", edad=35

el resultado será el siguiente mensaje en pantalla:

```
<Hola visitante soy el supervisor, tengo 35 años>.
```

Recuerde que se deben especificar en el mismo orden tanto el contenido de las variables a imprimir dentro de formato como las

variables en sí. La función printf() devuelve el número de caracteres realmente presentados en pantalla. Un valor negativo significa que se ha producido un error. Se pueden escribir enteros entre el signo de porcentaje y el caracter que especifica el tipo de dato a presentar. Esto sirve para determinar la longitud del campo, el número de decimales y un indicador de justificación a la izquierda.

Para especificar la longitud del campo, basta con escribir el número después del signo de porcentaje; después se agrega un punto y el número de posiciones decimales que se desea presentar en el caso de los números en coma flotante. Si la cadena es mayor que la anchura del campo, se truncan los caracteres por el final. Por ejemplo, %12.4f determina a un número de como máximo doce caracteres de longitud con cuatro posiciones para la parte decimal.

Cuando se aplica a cadenas de caracteres o enteros, el número después del punto determina la máxima longitud del campo. Por ejemplo %3.5s presenta una cadena que tiene al menos tres caracteres y que no excede de cinco. Si la cadena es mayor que el campo, se truncan los caracteres por el final. Por defecto, toda salida está justificada por la derecha. En otras palabras, si la anchura del campo es mayor que la de los datos presentados, estos son situados en la parte derecha del campo.

Puede forzar que la información quede justificada a la izquierda situando un signo menos (-) inmediatamente después del signo de porcentaje. Por ejemplo %-6.3f justifica un número en coma flotante por la izquierda con tres posiciones decimales en un campo de seis caracteres.

Ejemplo: Si saludo="hola" , printf("%-10s",saludo);  
presentará: <hola >

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

```
int scanf(char *formato,lista_arg)
```

Descripción

Esta función realiza la operación contraria a printf()es decir, lee datos de cualquier tipo desde el teclado hasta que se pulse un retorno de carro(<ENTER>). Sus parámetros también son similares a printf() ya que en formato se especifica el o los tipos de variables que se van a leer mientras que en lista\_arg se escriben las **direcciones de memoria (&)** de las variables. Por ejemplo:

```
scanf("%d",&edad); scanf("%i%c",&edad,&sexo);
```

La sección de formato corresponde a "%d" donde se indica que se va a leer un entero decimal; &edad corresponde a lista\_arg e indica que los caracteres leídos serán almacenados en la dirección que ocupa la variable edad. La diferencia entre la sintaxis de scanf() y printf() consiste en que en la lista de argumentos, scanf() necesita que se le especifique que el lugar donde vá a almacenar los datos es en la dirección de la variable( &edad). La única excepción es cuando se vá a leer una **cadena de caracteres** ya que este tipo de variables indican una **dirección por si mismas**. Ejemplo:

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

A continuación se presentan los códigos de formato de scanf().

Código	Interpretación
%c	Leer un único caracter
%d	Leer un entero decimal
%i	Leer un entero decimal
%e	Leer un número en coma flotante
%f	Leer un número en coma flotante
%h	Leer un entero corto
%o	Leer un número octal
%s	Leer una cadena de caracteres
%x	Leer un número hexadecimal
%p	Leer un puntero

Un espacio en blanco en la cadena de control da lugar a que scanf() salte uno o más espacios en blanco en el flujo de entrada. Un caracter blanco es un espacio, un tabulador o caracter de nueva línea. Por ejemplo, %d,%d dá lugar a que scanf() lea primero un entero, entonces lea y descarte la coma, y finalmente lea otro número. Si el caracter especificado no se encuentra, scanf() termina.

Un \* situado después del % y antes del código de formato lee los datos de tipo especificado pero elimina su asignación. Así, dada la entrada 10/20, el código scanf("%d\*%d",&x,&y); asigna el valor 10 a x, descarta el signo de división, y dá a y el valor 20.

Las órdenes de formato pueden especificar un modificador de máxima longitud de campo. Situado entre el % y el código de orden de formato, es un entero que limita la cantidad de caracteres a leer para cualquier campo. Por ejemplo, si se quieren leer sólo 15 caracteres en nombre se escribiría así: scanf("%15s",direccion);

Si el flujo de entrada fuera mayor de 15 caracteres, entonces una posterior llamada de entrada debería comenzar donde esta llamada la dejó. Por ejemplo, si Av.\_Corregidora\_#\_500

se ha introducido como respuesta al `scanf()` anterior, únicamente los primeros 15 caracteres (hasta la `a` de `corregidora`) serían situados en dirección debido al especificador de tamaño máximo. Cuando se hiciera otra llamada a `scanf()` tal como `scanf("%s",cadena);`  
`__#_500` se asignará a `cadena`.

## **FUNCIONES DE HORA, FECHA . OTRAS RELACIONES CON EL SISTEMA.**

```
#include <time.h>
    char *asctime(punt)
```

```
struct tm *punt
```

### Descripción:

Esta función regresa un puntero a una cadena que convierte la información almacenada en la estructura apuntada por `punt` de la forma siguiente:

dia mes horas:minutos:segundos año\n\n0

Por ejemplo:

Miercoles Jun 19 12:05:34 1999

El puntero a estructura pasado a `asctime()` se obtiene generalmente de `localtime()` o `gmtime()`. El buffer utilizado por `asctime()` para mantener la cadena de salida con formato se sitúa estáticamente en un array de caracteres y se sobrescribe cada vez que se llama a la función. Si se desea salvar el contenido de la cadena, es necesario copiarlo en otro lugar.

```
#include<time.h>
    clock_t clock()
```

### Descripción

Indica el tiempo empleado en la ejecución de un proceso.

Para transformar este valor en segundos, se divide entre el valor de la macro `CLK_TCK`. Se devuelve el valor `-1` si el tiempo no está disponible.

```
#include<time.h>
char *ctime(long *hora)
long *ctime(long *hora)
```

#### Descripción

Dado un puntero a la hora de calendario, la función `ctime()` convierte un tiempo almacenado como un valor de tipo `time_t` a una cadena de caracteres de la forma:

dia mes fecha horas:minutos:segundos año\n\n0

La hora de calendario se obtiene normalmente durante una llamada `atime()`. La función `ctime()` es equivalente a:

`asctime (localtime(hora))`

El buffer utilizado por `ctime()` para guardar la cadena de salida conformato se sitúa de forma estática en un array de caracteres y se sobrescribe cada vez que se llama a la función. Si se desea guardar el contenido de la cadena es necesario guardarla en otro lugar.

```
#include <time.h>
struct tm *localtime(time_t *hora)
```

#### Descripción

La función `localtime()` devuelve un puntero a la forma separada de hora en la estructura `tm`. La hora se representa con la hora local. Esta se obtiene normalmente a través de una llamada a `time()`.

La estructura utilizada por `localtime()` para mantener la hora separada, se situada de forma estática y se reescribe cada vez que se llama a la función. Si se desea guardar el contenido de la estructura, es necesario copiarla en otra variable.

```
#include<time.h>
time_t time(time_t hora)
```

#### Descripción

La función `time()` devuelve la hora actual del calendario del sistema. Si el sistema no tiene hora, devuelve -1.

Puede llamarse con un puntero nulo o con un puntero a una variable de tipo `time_t`. Si se utiliza este último, el argumento también es asignado a la hora de calendario.

## OTRAS FUNCIONES

```
include<stdlib.h>
void abort()
```

### Descripción

La función abort() dá lugar a la terminación automática del programa. Ningún archivo es volcado. En entornos que lo soportan, abort() devuelve un valor definido por la implementación al proceso que haya hecho la llamada (normalmente el sistema operativo). Su uso principal es prevenir una fuga del programa cerrando los archivos activos.

```
#include<stdlib.h>
void exit(int estado)
```

### Descripción

La función exit() da lugar inmediatamente a la terminación normal de un programa. El valor de estado se pasa al proceso de llamada - normalmente el sistema operativo- si el entorno lo soporta. Por convenio, el valor de estado es 0, cuando se ha producido una terminación normal del programa. Un valor distinto de cero puede utilizarse para indicar un error definido por la implementación.

```
#include<stdlib.h>
div_t div(int numer, int denom)
```

### Descripción

div() devuelve el cociente y el resto de la operación numer/denom. El tipo de estructura div\_t está definido en stdlib.h y tiene al menos estos dos campos:

```
int quot; /*el cociente*/
int rem /*el resto*/
```

```
#include<stdlib.h>
long labs(long num)
```

### Descripción

Al ejecutar esta función labs() se devuelve el valor absoluto de num, un long int.

```
#include <stdlib.h>
ldiv_t ldiv(long int numer, long int denom)
```

#### Descripción

Devuelve el cociente y el resto de la operación numer/denom que son números enteros de tipo long.

El tipo de estructura ldiv\_t está definido en stdlib.h y tiene al menos estos dos campos:

```
int quot; /*el cociente*/
int rem /*el resto*/
```

```
#include <stdlib.h>
int rand()
```

#### Descripción

rand() genera un flujo de números pseudoaleatorios. Cada vez que se llama, se devuelve un entero entre 0 y RAND\_MAX.

```
#include <stdlib.h>
void srand(unsigned int valor)
```

#### Descripción

La función srand() utiliza valor para fijar un punto de partida para el flujo generado por rand(), que devuelve números pseudoaleatorios.

La función srand() se utiliza normalmente para permitir que ejecuciones múltiples de un programa utilicen diferentes flujos de números pseudoaleatorios.

```
#include <stdlib.h>
int system(char *cad)
```

#### Descripción

La función system() pasa la cadena apuntada por cad como una orden al procesador de órdenes del sistema operativo.

El system() se llama con un putero nulo, devuelve un valor distinto de cero si está presente un procesador de órdenes; en cualquier otro caso devuelve 0. Esta función devuelve cero si fue completamente ejecutada; en cualquier otro caso devuelve un valor distinto de cero.