# Librerías ANSI C

Librerías estándar C


Steven R. Davidson steven@conclase.net y
Salvador Pozo salvador@conclase.net

Con Clase: http://www.conclase.net

# Librerías ANSI C

Hemos tomado la decisión de separar las descripciones de las librerías de C y C++ de los cursos.

Por una parte, porque creemos que quedará más claro que C/C++ es un lenguaje completo en si mismo, las librerías ANSI que se incluyen con todos los compiladores están escritas en C o en ensamblador, y por lo tanto no son "imprescindibles" para escribir programas en C.

Es cierto que son una gran ayuda, pero también lo son las librerías que escribe cada uno para su uso personal.

Por otra parte, estas páginas podrán usarse como consulta para ver el funcionamiento de cada función individual, sin necesidad de buscarlas a través del curso. Para que la consulta sea más fácil, se incluye un índice alfabético de funciones, y un índice de ficheros de cabecera.

Hay que mencionar que todos los ejemplos y explicaciones se refieren a C estándar. De todos modos, si se quieren utilizar estas librerías en C++ no hay ningún inconveniente, pero para ceñirse al estándar C++ los ficheros de cabecera se forman sin la extensión ".h" y añadiendo el prefijo "c" al nombre. Por ejemplo, el fichero de cabecera para stdio en C es <stdio.h>, y en C++ es <cstdio>.

*Nota: algunas descripciones de funciones, estructuras y macros han sido extraídas de la ayuda de los compiladores de Borland y del libro: "C How to Program" de H.M. DEITEL & P.J. DEITEL.*

# Librería assert ANSI C

Únicamente define la macro de depuración assert.

## Macros

assert

# Librería ctype ANSI C

Contiene los prototipos de las funciones y macros de clasificación de caracteres.

## Tabla de referencia rápida:

| Función | Valores |
|---|---|
| isalnum | (A - Z o a - z) o (0 - 9) |
| isalpha | (A - Z o a - z) |
| isascii | 0 - 127 (0x00-0x7F) |
| iscntrl | (0x7F o 0x00-0x1F) |
| isdigit | (0 - 9) |
| isgraph | Imprimibles menos ' ' |
| islower | (a - z) |
| isprint | Imprimibles incluido ' ' |
| ispunct | Signos de puntuación |
| isspace | espacio, tab, retorno de línea, cambio de línea, tab vertical, salto de página (0x09 a 0x0D, 0x20). |
| isupper | (A-Z) |
| isxdigit | (0 to 9, A to F, a to f) |

## Funciones

tolower toupper

## Macros

isalnum isalpha isascii iscntrl
isdigit isgraph islower isprint
ispunct isspace isupper isxdigit
toascii

# Librería errno ANSI C

Define constantes para los códigos de error, puedes consultarlos editando el fichero errno.h.

## Macros

errno

# Librería float ANSI C

Contiene parámetros de entorno, información sobre limitaciones y rangos para tipos reales.

Nota: Algunas macros serán definidas de igual o mayor en magnitud (valor absoluto) a los valores indicados, a excepción de la macro FLT_ROUNDS.

## Macros

| | | | |
|---|---|---|---|
| DBL_DIG | DBL_EPSILON | DBL_MANT_DIG | DBL_MAX |
| DBL_MAX_10_EXP | DBL_MAX_EXP | DBL_MIN | DBL_MIN_10_EXP |
| DBL_MIN_EXP | FLT_DIG | FLT_EPSILON | FLT_MANT_DIG |
| FLT_MAX | FLT_MAX_10_EXP | FLT_MAX_EXP | FLT_MIN |
| FLT_MIN_10_EXP | FLT_MIN_EXP | FLT_RADIX | FLT_ROUNDS |
| LDBL_DIG | LDBL_EPSILON | LDBL_MANT_DIG | LDBL_MAX |
| LDBL_MAX_10_EXP | LDBL_MAX_EXP | LDBL_MIN | LDBL_MIN_10_EXP |
| LDBL_MIN_EXP | | | |

# Fichero limits ANSI C

Contiene parámetros de entorno, información sobre limitaciones y rangos para tipos enteros.

## Constantes:

| Constante | Significado |
|---|---|
| CHAR_BIT | Número de bits del tipo char |
| CHAR_MIN | Valor mínimo del tipo char |
| CHAR_MAX | Valor máximo del tipo char |
| INT_MIN | Valor mínimo del tipo int |
| INT_MAX | Valor máximo del tipo int |
| LONG_MIN | Valor mínimo del tipo long |
| LONG_MAX | Valor máximo del tipo long |
| SCHAR_MIN | Valor mínimo del tipo char con signo |
| SCHAR_MAX | Valor máximo del tipo char con signo |
| SHRT_MIN | Valor mínimo del tipo short |
| SHRT_MAX | Valor máximo del tipo short |
| UCHAR_MAX | Valor máximo de unsigned char |
| USHRT_MAX | Valor máximo unsigned short |
| UINT_MAX | Valor máximo unsigned int |
| ULONG_MAX | Valor máximo unsigned long |

# Librería locale ANSI C

Contiene los prototipos de las funciones, macros, y tipos para manipular y controlar varias opciones pertenecientes a la localidad del sistema.

## Funciones

localeconv setlocale

## Macros

LC_ALL LC_COLLATE LC_CTYPE LC_MONETARY
LC_NUMERIC LC_TIME NULL

## Estructuras

lconv

# Librería math ANSI C

Contiene los prototipos de las funciones y otras definiciones para el uso y manipulación de funciones matemáticas.

## Funciones

| | | | |
|---|---|---|---|
| acos | asin | atan | atan2 |
| ceil | cos | cosh | exp |
| fabs | floor | fmod | frexp |
| ldexp | log | log10 | modf |
| pow | sin | sinh | sqrt |
| tan | tanh | | |

## Macros

HUGE_VAL

# Librería setjmp ANSI C

Contiene los prototipos para las funciones y un tipo para crear y manipular el entorno al hacer llamadas: registros, pilas, etc..

## Funciones

longjmp setjmp

## Estructuras

jmp_buf

# Librería signal ANSI C

Contiene las funciones, macros, y tipos para crear y manipular señales del sistema.

## Funciones

raise signal

## Macros

SIGABRT SIGFPE SIGILL SIGSEGV
SIGTERM SIG_DFL SIG_ERR SIG_IGN

## Estructuras

sig_atomic_t

# Librería stdarg ANSI C

Contiene las macros y tipos para crear y manipular argumentos de variables.

## Macros

va_arg va_end va_list va_start

# Librería stddef ANSI C

Contiene las macros, y tipos comunes.

## Macros

NULL offsetof

## Estructuras

ptrdiff_t size_t wchar_t

# Librería stdio ANSI C

Contiene los prototipos de las funciones, macros, y tipos para manipular datos de entrada y salida.

## Funciones

| | | | |
|---|---|---|---|
| clearerr | fclose | feof | ferror |
| fflush | fgetc | fgetpos | fgets |
| fopen | formato | fprintf | fputc |
| fputs | fread | freopen | fscanf |
| fseek | fsetpos | ftell | fwrite |
| getc | getchar | gets | perror |
| printf | putc | putchar | puts |
| remove | rename | rewind | scanf |
| setbuf | setvbuf | sprintf | sscanf |
| tmpfile | tmpnam | ungetc | vfprintf |
| vprintf | vsprintf | | |

## Macros

| | | | |
|---|---|---|---|
| BUFSIZ | EOF | FILENAME_MAX | FOPEN_MAX |
| L_tmpnam | NULL | SEEK_CUR | SEEK_END |
| SEEK_SET | stderr | stdin | stdout |
| TMP_MAX | _IOFBF | _IOLBF | _IONBF |

## Estructuras

| | | |
|---|---|---|
| FILE | fpos_t | size_t |

# Librería stdlib ANSI C

Contiene los prototipos de las funciones, macros, y tipos para utilidades de uso general.

## Funciones

| | | | |
|---|---|---|---|
| abort | abs | atexit | atof |
| atoi | atol | bsearch | calloc |
| div | exit | free | getenv |
| labs | ldiv | malloc | mblen |
| mbstowcs | mbtowc | qsort | rand |
| realloc | srand | strtod | strtol |
| strtoul | system | wctomb | |

## Macros

EXIT_FAILURE EXIT_SUCCESS MB_CUR_MAX NULL RAND_MAX

## Estructuras

| | | | |
|---|---|---|---|
| div_t | ldiv_t | size_t | wchar_t |

# Librería string ANSI C

Contiene los prototipos de las funciones y macros de clasificación de caracteres.

## Funciones

| | | | |
|---|---|---|---|
| memchr | memcmp | memcpy | memmove |
| memset | strcat | strchr | strcmp |
| strcoll | strcpy | strcspn | strerror |
| strlen | strncat | strncmp | strncpy |
| strpbrk | strrchr | strspn | strstr |
| strtok | strxfrm | | |

## Macros

NULL

## Estructuras

size_t

# Librería time ANSI C

Contiene los prototipos de las funciones, macros, y tipos para manipular la hora y la fecha del sistema.

## Funciones

asctime clock ctime difftime
gmtime localtime mktime strftime
time

## Macros

CLOCKS_PER_SEC NULL

## Estructuras

clock_t size_t time_t tm

# Indice de funciones

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## -A-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| abort | stdlib | stdlib.h | cstdlib |
| abs | stdlib | stdlib.h | cstdlib |
| acos | math | math.h | cmath |
| asctime | time | time.h | ctime |
| asin | math | math.h | cmath |
| atan | math | math.h | cmath |
| atan2 | math | math.h | cmath |
| atexit | stdlib | stdlib.h | cstdlib |
| atof | stdlib | stdlib.h | cstdlib |
| atoi | stdlib | stdlib.h | cstdlib |
| atol | stdlib | stdlib.h | cstdlib |

## -B-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| bsearch | stdlib | stdlib.h | cstdlib |

## -C-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| calloc | stdlib | stdlib.h | cstdlib |
| ceil | math | math.h | cmath |
| clearerr | stdio | stdio.h | cstdio |
| clock | time | time.h | ctime |
| cos | math | math.h | cmath |
| cosh | math | math.h | cmath |

| | | | |
|---|---|---|---|
| ctime | time | time.h | ctime |

## -D-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| difftime | time | time.h | ctime |
| div | stdlib | stdlib.h | cstdlib |

## -E-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| exit | stdlib | stdlib.h | cstdlib |
| exp | math | math.h | cmath |

## -F-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| fabs | math | math.h | cmath |
| fclose | stdio | stdio.h | cstdio |
| feof | stdio | stdio.h | cstdio |
| ferror | stdio | stdio.h | cstdio |
| fflush | stdio | stdio.h | cstdio |
| fgetc | stdio | stdio.h | cstdio |
| fgetpos | stdio | stdio.h | cstdio |
| fgets | stdio | stdio.h | cstdio |
| floor | math | math.h | cmath |
| fmod | math | math.h | cmath |
| fopen | stdio | stdio.h | cstdio |
| formato | stdio | stdio.h | cstdio |
| fprintf | stdio | stdio.h | cstdio |
| fputc | stdio | stdio.h | cstdio |
| fputs | stdio | stdio.h | cstdio |
| fread | stdio | stdio.h | cstdio |

| | | | |
|---|---|---|---|
| free | stdlib | stdlib.h | cstdlib |
| freopen | stdio | stdio.h | cstdio |
| frexp | math | math.h | cmath |
| fscanf | stdio | stdio.h | cstdio |
| fseek | stdio | stdio.h | cstdio |
| fsetpos | stdio | stdio.h | cstdio |
| ftell | stdio | stdio.h | cstdio |
| fwrite | stdio | stdio.h | cstdio |

## -G-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| getc | stdio | stdio.h | cstdio |
| getchar | stdio | stdio.h | cstdio |
| getenv | stdlib | stdlib.h | cstdlib |
| gets | stdio | stdio.h | cstdio |
| gmtime | time | time.h | ctime |

## -L-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| labs | stdlib | stdlib.h | cstdlib |
| ldexp | math | math.h | cmath |
| ldiv | stdlib | stdlib.h | cstdlib |
| localeconv | locale | locale.h | clocale |
| localtime | time | time.h | ctime |
| log | math | math.h | cmath |
| log10 | math | math.h | cmath |
| longjmp | setjmp | setjmp.h | csetjmp |

## -M-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|

| | | | |
|---|---|---|---|
| malloc | stdlib | stdlib.h | cstdlib |
| mblen | stdlib | stdlib.h | cstdlib |
| mbstowcs | stdlib | stdlib.h | cstdlib |
| mbtowc | stdlib | stdlib.h | cstdlib |
| memchr | string | string.h | cstring |
| memcmp | string | string.h | cstring |
| memcpy | string | string.h | cstring |
| memmove | string | string.h | cstring |
| memset | string | string.h | cstring |
| mktime | time | time.h | ctime |
| modf | math | math.h | cmath |

## -P-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| perror | stdio | stdio.h | cstdio |
| pow | math | math.h | cmath |
| printf | stdio | stdio.h | cstdio |
| putc | stdio | stdio.h | cstdio |
| putchar | stdio | stdio.h | cstdio |
| puts | stdio | stdio.h | cstdio |

## -Q-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| qsort | stdlib | stdlib.h | cstdlib |

## -R-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| raise | signal | signal.h | csignal |
| rand | stdlib | stdlib.h | cstdlib |
| realloc | stdlib | stdlib.h | cstdlib |

| | | | |
|---|---|---|---|
| remove | stdio | stdio.h | cstdio |
| rename | stdio | stdio.h | cstdio |
| rewind | stdio | stdio.h | cstdio |

## - S -

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| scanf | stdio | stdio.h | cstdio |
| setbuf | stdio | stdio.h | cstdio |
| setjmp | setjmp | setjmp.h | csetjmp |
| setlocale | locale | locale.h | clocale |
| setvbuf | stdio | stdio.h | cstdio |
| signal | signal | signal.h | csignal |
| sin | math | math.h | cmath |
| sinh | math | math.h | cmath |
| sprintf | stdio | stdio.h | cstdio |
| sqrt | math | math.h | cmath |
| srand | stdlib | stdlib.h | cstdlib |
| sscanf | stdio | stdio.h | cstdio |
| strcat | string | string.h | cstring |
| strchr | string | string.h | cstring |
| strcmp | string | string.h | cstring |
| strcoll | string | string.h | cstring |
| strcpy | string | string.h | cstring |
| strcspn | string | string.h | cstring |
| strerror | string | string.h | cstring |
| strftime | time | time.h | ctime |
| strlen | string | string.h | cstring |
| strncat | string | string.h | cstring |
| strncmp | string | string.h | cstring |
| strncpy | string | string.h | cstring |
| strpbrk | string | string.h | cstring |
| strrchr | string | string.h | cstring |

| | | | |
|---|---|---|---|
| strspn | string | string.h | cstring |
| strstr | string | string.h | cstring |
| strtod | stdlib | stdlib.h | cstdlib |
| strtok | string | string.h | cstring |
| strtol | stdlib | stdlib.h | cstdlib |
| strtoul | stdlib | stdlib.h | cstdlib |
| strxfrm | string | string.h | cstring |
| system | stdlib | stdlib.h | cstdlib |

## -T-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| tan | math | math.h | cmath |
| tanh | math | math.h | cmath |
| time | time | time.h | ctime |
| tmpfile | stdio | stdio.h | cstdio |
| tmpnam | stdio | stdio.h | cstdio |
| tolower | ctype | ctype.h | cctype |
| toupper | ctype | ctype.h | cctype |

## -U-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| ungetc | stdio | stdio.h | cstdio |

## -V-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| vfprintf | stdio | stdio.h | cstdio |
| vprintf | stdio | stdio.h | cstdio |
| vsprintf | stdio | stdio.h | cstdio |

## -W-

| Función | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| wctomb | stdlib | stdlib.h | cstdlib |

# Función abort ANSI C

**Librería: stdlib**

```
void abort(void);
```

Ocasiona una terminación abnormal del programa, al menos que la señal SIGABRT está siendo capturado y un controlador de señales no regresa. Si streams abiertos de salida son despejados o streams abiertos son cerrados o ficheros temporales son borrados es cosa de la definición de la implementación. Una forma definida de la implementación del estado "terminación sin éxito" es retornado al entorno local por medio de la llamada a la función **raise(SIGABRT)**. La función *abort* no puede regresar a su invocador.

## Valor de retorno:

La función *abort* no retorna ningún valor.

## Ejemplo:

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

int main( void )
{
   puts( "Introduzca un caracter. Para salir, escriba \'q\':" );

   while( 1 )   if( toupper(getchar()) == 'Q' )   abort();

   return 0;
}
```

# Función abs ANSI C

**Librería: stdlib**

```
int abs(int num);
```

Calcula el valor absoluto de un entero **num**. Si el resultado no puede ser representado, el comportamiento no está definido.

## Valor de retorno:

La función *abs* retorna el valor absoluto.

## Ejemplo:

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

int main()
{
   int num;

   puts( "Escriba un numero entero:" );
   scanf( "%d", &num );
   printf( "abs( %d ) = %d\n", num, abs(num) );

   return 0;
}
```

# Funciónn acos ANSI C

**Librería: math**

```
double acos(double x);
```

Calcula el valor principal del arco coseno de **x**. Puede producirse un error de dominio para los argumentos que no estén en el intervalo [-1, +1].

## Valor de retorno:

La función acos retorna el arco coseno en el intervalo [0, PI] radianes.

## Ejemplo:

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

int main()
{
   double x = 0.2345;

   printf( "acos( %f ) = %f\n", x, acos(x) );
   return 0;
}
```

# Función asctime ANSI C

**Librería: time**

```
char *asctime(const struct tm *tiempoPtr);
```

La función *asctime* convierte el tiempo en formato separado en la estructura apuntada por **tiempoPtr** en una cadena en la forma: **Tue May 15 19:07.04 2001\n\0**

## Valor de retorno:

La función *asctime* retorna un puntero a la cadena.

## Ejemplo:

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

int main( void )
{
   long int i=0;
   time_t comienzo, final;
   struct tm *tiempoComienzoPtr, *tiempoFinalPtr;

   comienzo = time( NULL );
   for( i=0; i<10000; i++ )   printf( "-" );
   final = time( NULL );

   printf( "Comienzo: %u s\n", comienzo );
   printf( "Final: %u s\n", final );
   printf( "Número de segundos transcurridos desde el comienzo del programa: %f s\n",
difftime(final, comienzo) );

   tiempoComienzoPtr = gmtime( &comienzo );
   tiempoFinalPtr = gmtime( &final );
   printf( "Comienzo: %s\n", asctime(tiempoComienzoPtr) );
   printf( "Final: %s\n", asctime(tiempoFinalPtr) );

   return 0;
}
```

# Función asin ANSI C

**Librería: math**

```
double asin(double x);
```

Calcula el valor principal del arco seno de **x**. Puede producirse un error de dominio para los argumentos que no estén en el intervalo [-1, +1].

## Valor de retorno:

La función asin retorna el arco seno en el intervalo [-PI/2, +PI/2] radianes.

## Ejemplo:

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

int main()
{
   double x = 0.2345;

   printf( "asin( %f ) = %f\n", x, asin(x) );
   return 0;
}
```

# Función atan ANSI C

**Librería: math**

```
double atan(double x);
```

Calcula el valor principal del arco tangente de **x**.

## Valor de retorno:

La función atan retorna el arco tangente en el intervalo [-PI/2, +PI/2] radianes.

## Ejemplo:

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

int main()
{
   double x = 0.2345;

   printf( "atan( %f ) = %f\n", x, atan(x) );
   return 0;
}
```

# Función atan2 ANSI C

**Librería: math**

```
double atan2(double y, double x);
```

Calcula el valor principal del arco tangente de **y/x**, usando los signos de ambos argumentos para determinar el cuadrante del valor de retorno. Puede producirse un error de dominio si ambos argumentos son cero.

## Valor de retorno:

La función atan2 retorna el arco tangente de **y/x**, en el intervalo [-PI, +PI] radianes.

## Ejemplo:

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

int main()
{
   double x = 0.2345, y = -3.4567;

   printf( "atan2( %f, %f ) = %f\n", y, x, atan2(y,x) );
   return 0;
}
```

# Función atexit ANSI C

**Librería: stdlib**

```
int atexit(void (*func)(void));
```

Registra la función apuntada por **func**, para ser llamada sin argumentos al momento de terminación normal del programa. La implementación deberá aceptar el registro de al menos 32 funciones.

## Valor de retorno:

La función *atexit* retorna cero si el registro tiene éxito, un valor distinto a cero si falla.

## Ejemplo:

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

void antes_de_salir(void)
{
   puts( "Nos vamos..." );
}

int main( void )
{
   atexit( antes_de_salir );

   puts( "Esto es una prueba. Introduzca cualquier letra para terminar." );
   getchar();

   return 0;
}
```

# Función atof ANSI C

**Librería: stdlib**

```
double atof(const char *numPtr);
```

Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **double**.

## Valor de retorno:

La función *atof* retorna el valor convertido.

## Ejemplo:

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

int main()
{
   char numPtr[11] = "123.456789";

   printf( "Convirtiendo la cadena \"%s\" en un numero: %f\n", numPtr, atof(numPtr)
);

   return 0;
}
```

# Función atoi ANSI C

**Librería: stdlib**

```
int atoi(const char *numPtr);
```

Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **int**.

## Valor de retorno:

La función *atoi* retorna el valor convertido.

## Ejemplo:

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

int main()
{
   char numPtr[5] = "1234";

   printf( "Convirtiendo la cadena \"%s\" en un numero: %d\n", numPtr, atoi(numPtr)
);

   return 0;
}
```

# Función atol ANSI C

**Librería: stdlib**

```
long int atol(const char *numPtr);
```

Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **long**.

## Valor de retorno:

La función *atol* retorna el valor convertido.

## Ejemplo:

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

int main()
{
   char numPtr[11] = "1234567890";

   printf( "Convirtiendo la cadena \"%s\" en un numero: %u\n", numPtr, atol(numPtr)
);

   return 0;
}
```

# Función bsearch ANSI C

**Librería: stdlib**

```
void *bsearch(const void *clave, const void *base,
   size_t nmemb, size_t tamanyo,
   int (*comparar)(const void *, const void *));
```

Busca un array de **nmemb** objetos, el elemento inicial del que es apuntado por **base**, para un elemento que empareje el objeto apuntado por **clave**. El tamaño de cada elemento del array está especificado por **tamanyo**. La función de la comparación apuntada por **comparar** es llamada con dos argumentos que apuntan al objeto **clave** y a un elemento del array, en ese orden. La función retornará un entero menor, igual, o mayor que cero si el objeto **clave** es considerado, respectivamente a ser menor, igual, o mayor que el elemento del array. El array consistirá de: todos los elementos que comparan a ser menores, iguales, o mayores que el objeto **clave**, en ese orden.

## Valor de retorno:

La función *bsearch* retorna un puntero a un elemento emparejado del array, o a un puntero nulo si ninguna pareja es encontrada. Si dos elementos comparados son iguales, el elemento emparejado no es especificado.

## Ejemplo:

```
/* EJEMPLO */

/* Implementación del algoritmo quick-sort para la ordenación de vectores
** y bsearch para buscar ciertos elementos
*/

#include <stdlib.h>
#include <stdio.h>

#define ELEMENTOS 100

/* Prototipo de la función de comparar */
int comparar(const void *arg1, const void *arg2);
{
 if(*(int *)arg1 < *(int *)arg2) return -1;
   else if(*(int *)arg1 > *(int *)arg2) return 1;
   else return 0;
}

int main()
{
   int i, num;
   int lista[ELEMENTOS], int *elementoPtr;

   /* Contenido aleatorio */
   for(i = 0; i < ELEMENTOS; i++) lista[i] = rand() % 100 + 1;

   /* Quick-Sort */
   qsort(lista, ELEMENTOS, sizeof(lista[0]), comparar);

    /* Mostramos la lista ordenada */
   for(i = 0; i < ELEMENTOS; i++) printf("%d ", lista[i]);
   printf("\n");

   /* Ahora busquemos algunos elementos en la lista */
```

```
   puts( "Especifique un numero a encontrar dentro de la lista ordenada\n(un numero
negativo para salir):" );
   scanf( "%d", &num );
   while( num >= 0 )
   {
      if( (elementoPtr = bsearch( num, lista, ELEMENTOS, sizeof(lista[0]), comparar
)) != NULL )
          printf( "Encontrado: %d\n", *elementoPtr );
      else
          printf( "No encontrado: %d\n", num );
      scanf( "%d", &num );
   }

   return 0;
}
/* FIN DE EJEMPLO */
```

# Función calloc ANSI C

**Librería: stdlib**

```
void *calloc(size_t nmemb, size_t tamanyo);
```

Adjudica espacio para un array de **nmemb** objetos, cada cual tiene como tamaño **tamanyo**. El espacio es inicializado a cero todos los bits.

## Valor de retorno:

La función *calloc* retorna o bien un puntero nulo o bien un puntero al espacio adjudicado.

## Ejemplo:

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

int main( void )
{
   int *numPtr, i;
   size_t tamanyo=0;

   printf( "Introduzca el tamaño de la lista: " );
   scanf( "%d", &tamanyo );

   puts( "Adjudicamos espacio a la lista (con calloc)." );
   numPtr = (int *)calloc( tamanyo, sizeof(int) );

   for( i=0; i<tamanyo-1; i++ )
      printf( "%d, ", numPtr[i] = rand() % 100 + 1 );
   printf( "%d\n", numPtr[i] = rand() % 100 + 1 );

   numPtr = (int *)realloc( numPtr, tamanyo/=2 );
   printf( "Recortamos la lista a la mitad: %d\n", tamanyo );
   for( i=0; i<tamanyo-1; i++ )
      printf( "%d, ", numPtr[i] );
   printf( "%d\n", numPtr[i] );

   puts( "Liberamos el espacio (con realloc)." );
   realloc( numPtr, 0 );    /* Una forma poco ortodoxa, pero válida, de liberar
espacio */

   return 0;
}
```

# Función ceil ANSI C

**Librería: math**

```
double ceil(double x);
```

Calcula el valor integral más pequeño que no sea menor de **x**.

## Valor de retorno:

La función ceil retorna el resultado de la función "techo" de **x**.

## Ejemplo:

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

int main()
{
   double x = 6.54321;

   printf( "ceil( %f ) = %f\n", x, ceil(x) );
   return 0;
}
```

# Función clearerr ANSI C

**Librería: stdio**

```
void clearerr(FILE *stream);
```

La función *clearerr* despeja los indicadores de final de fichero y de posición de fichero para el stream apuntado por **stream** al comienzo del fichero.

## Valor de retorno:

La función *clearerr* no retorna ningún valor.

## Ejemplo:

```
#include <stdio.h>

int main( void )
{
   char c='z';
   char nombre[11] = "datos2.dat";
   FILE *fichero;

   if( NULL == (fichero = fopen(nombre, "r")) )
      printf( "ERROR: No se pudo abrir el fichero, %s\n", fichero );

   if( EOF == fputc( c, fichero ) )    printf( "fputc => Error\n" );    /* Creamos un
error */
   else   printf( "Caracter: %c, se ha escrito en el fichero: %s\n", c, nombre );
/* No se dará el caso */

   if( ferror(fichero) )
   {
      printf( "Se han encontrado errores al realizar una operacion con el fichero,
%s.\n", nombre );
      clearerr( fichero );
      printf( "Los errores han sido despejado para el stream.\n" );
   }
   else    printf( "No hubo errores con el fichero: %s\n", nombre );    /* Tampoco se
dará el caso */

   fclose( fichero );

   return 0;
}
```

# Función clock ANSI C

**Librería: time**

```
clock_t clock(void);
```

La función *clock* determina el tiempo usado del procesador.

## Valor de retorno:

La función *clock* retorna la mejor aproximación por la implementación del tiempo del procesador usado por el programa desde el comienzo de un período, definido según la implementación, relacionado solamente a la invocación del programa. Para determinar el tiempo en segundos, el valor retornado por la función *clock* debería ser dividido por el valor de la macro CLOCKS_PER_SEC. Si el tiempo usado del procesador no está disponible o su valor no puede ser representado, la función retorna el valor **(clock_t)-1**.

## Ejemplo:

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

int main( void )
{
   long int i=0;
   clock_t comienzo;

   comienzo=clock();
   for( i=0; i<10000; i++ )   printf( "-" );

   printf( "Número de segundos transcurridos desde el comienzo del programa: %f s\n",
(clock()-comienzo)/(double)CLOCKS_PER_SEC );

   return 0;
}
```

# Función cos ANSI C

## Librería: math

```
double cos(double x);
```

Calcula el coseno de **x** (medido en radianes).

## Valor de retorno:

La función cos retorna el coseno, en radianes.

## Ejemplo:

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

int main()
{
   double x = 3.1416/3.0;

   printf( "cos( %f ) = %f\n", x, cos(x) );
   return 0;
}
```

# Función cosh ANSI C

**Librería: math**

```
double cosh(double x);
```

Calcula el coseno hiperbólico de **x**. Puede producirse un error de recorrido si la magnitud de **x** es demasiada grande.

## Valor de retorno:

La función cosh retorna el coseno hiperbólico.

## Ejemplo:

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

int main()
{
   double x = 3.0;

   printf( "cosh( %f ) = %f\n", x, cosh(x) );
   return 0;
}
```

# Función ctime ANSI C

**Librería: time**

```
char *ctime(const time_t *tiempoPtr);
```

La función *ctime* convierte el tiempo en formato condensado apuntado por **tiempoPtr** en el tiempo local cadena en la forma de una cadena. Esto es equivalente a: **asctime(localtime(tiempoPtr));**

## Valor de retorno:

La función *ctime* retorna el puntero retornado por la función asctime con el tiempo en formato separado como argumento.

## Ejemplo:

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

int main( void )
{
   long int i=0;
   time_t comienzo, final;
   struct tm *tiempoComienzoPtr, *tiempoFinalPtr;

   comienzo = time( NULL );
   for( i=0; i<10000; i++ )   printf( "-" );
   final = time( NULL );

   printf( "Comienzo: %u s\n", comienzo );
   printf( "Final: %u s\n", final );
   printf( "Número de segundos transcurridos desde el comienzo del programa: %f s\n",
difftime(final, comienzo) );

   tiempoComienzoPtr = gmtime( &comienzo );
   tiempoFinalPtr = gmtime( &final );
   printf( "Comienzo: %s\n", asctime(tiempoComienzoPtr) );
   printf( "Final: %s\n", asctime(tiempoFinalPtr) );

   return 0;
}
```

# Función difftime ANSI C

**Librería: time**

```
double difftime(time_t tiempo1, time_t tiempo0);
```

La función *difftime* calcula la diferencia entre dos tiempos en formato condensado: **tiempo1 - tiempo0**.

## Valor de retorno:

La función *difftime* retorna la diferencia expresada en segundos como **double**.

## Ejemplo:

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

int main( void )
{
   long int i=0;
   time_t comienzo, final;

   comienzo = time( NULL );
   for( i=0; i<10000; i++ )   printf( "-" );
   final = time( NULL );

   printf( "Comienzo: %u s\n", comienzo );
   printf( "Final: %u s\n", final );
   printf( "Número de segundos transcurridos desde el comienzo del programa: %f s\n",
difftime(final, comienzo) );

   return 0;
}
```

# Función div ANSI C

**Librería: stdlib**

```
div_t div(int num, int denom);
```

Calcula el cociente y resto de la división del numerador **num** entre el denominador **denom**. Si la división es inexacta, el cociente resultante es el entero de menor magnitud que es el más próximo al cociente algebraico. Si el resultado no puede ser representado, el comportamiento no está definido; de lo contrario, **quot * denom + rem** igualará **num**.

## Valor de retorno:

La función *div* retorna la estructura de tipo div_t, conteniendo el cociente y el resto. La estructura contiene los siguientes miembros, en cualquier orden:

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

## Ejemplo:

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

int main()
{
   div_t d;
   int num, denom;

   puts( "Escriba el numerador y el denominador (separados por un espacio):" );
   scanf( "%d %d", &num, &denom );
   d = div( num, denom );
   printf( "ldiv( %d, %d ) : cociente = %d, resto = %d\n", num, denom, d.quot, d.rem
);

   return 0;
}
```

# Función exit ANSI C

**Librería: stdlib**

```
void exit(int estado);
```

Ocasiona una terminación normal del programa. Si se ejecuta más de una llamada de la función *exit*, el comportamiento no está definido. Primeramente, todas las funciones registradas por la función atexit son llamadas, en el orden inverso de sus registros. Cada función es llamada tantas veces como fue resgistrada. Acto seguido, todos los streams abiertos con datos almacenados aún sin escribir son despejados, todos los streams abiertos son cerrados, y todos los ficheros creados por la función tmpfile son borrados.

Finalmente, el control es regresado al entorno local. Si el valor de **estado** es cero o EXIT_SUCCESS, una forma definida según la implementación del estado "terminación con éxito" es retornada. Si el valor de **estado** es EXIT_FAILURE, una forma definida según la implementación del estado "terminación sin éxito" es retornada. De lo contrario el estado retornado está definida según la implementación. La función *exit* no puede regresar a su invocador.

## Valor de retorno:

La función *exit* no retorna ningún valor.

## Ejemplo:

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

int main( void )
{
   puts( "Introduzca un caracter. Para salir, escriba \'q\':" );

   while( 1 )   if( toupper(getchar()) == 'Q' )   exit(0);

   return 0;
}
```

# Función exp ANSI C

**Librería: math**

```
double exp(double x);
```

Calcula la función exponencial de **x**.

## Valor de retorno:

La función exp retorna el valor de $e^x$.

## Ejemplo:

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

int main()
{
   double x = -5.567;

   printf( "exp( %f ) = %f\n", x, exp(x) );
   return 0;
}
```

# Función fabs ANSI C

**Librería: math**

```
double fabs(double x);
```

Calcula el valor absoluto del número de coma flotante, **x**.

## Valor de retorno:

La función fabs retorna el valor absoluto de **x**.

## Ejemplo:

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

int main()
{
   double x = -6.54321;

   printf( "fabs( %f ) = %f\n", x, fabs(x) );
   return 0;
}
```

# Función fclose ANSI C

**Librería: stdio**

```
int fclose(FILE *stream);
```

El stream apuntado por **stream** será despejado y el fichero asociado, cerrado. Cualquier dato almacenado aún sin escribir para el stream será enviado al entorno local para ser escritos al fichero; cualquier dato almacenado aún sin leer será descartado. El stream es desasociado del fichero. Si el almacenamiento asociado fue automáticamente adjudicado, será desadjudicado.

## Valor de retorno:

La función *fclose* retorna cero si el stream fue cerrado con éxito. Si se detectaron errores, entonces retorna EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   FILE *fichero;
   char nombre[10] = "datos.dat";

   fichero = fopen( nombre, "w" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   if( !fclose(fichero) )
      printf( "Fichero cerrado\n" );
   else
   {
      printf( "Error: fichero NO CERRADO\n" );
      return 1;
```

```
    }

    return 0;
}
```

# Función feof ANSI C

**Librería: stdio**

```
int feof(FILE *stream);
```

La función *feof* comprueba el indicador de final de fichero para el stream apuntado por **stream**.

## Valor de retorno:

La función *feof* retorna un valor distinto a cero si y sólo si el indicador de final de fichero está activado para **stream**.

## Ejemplo:

```
#include <stdio.h>

int main( void )
{
   int tamanyo=0;
   FILE *ficheroEntrada, *ficheroSaliada;
   char nombreEntrada[11]="datos2.dat", nombreSalida[11]="datos3.dat";

   ficheroEntrada = fopen( nombreEntrada, "r" );
   printf( "Fichero de Lectura: %s -> ", nombreEntrada );
   if( ficheroEntrada )
      printf( "existe (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }
   ficheroSalida = fopen( nombreSalida, "w" );
   printf( "Fichero de Lectura: %s -> ", nombreSalida );
   if( ficheroSalida )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   while( !feof(ficheroEntrada) )
   {
      fputc( fgetc(ficheroEntrada)+3, ficheroSalida );    /* Desplazar cada carácter
3 caracteres: a -> d, b -> e, ... */
      tamanyo++;
   }
   printf( "El fichero \'%s\' contiene %d caracteres.\n", nombreEntrada, tamanyo );

   if( !fclose(ficheroSalida) )
      printf( "Fichero: %s cerrado\n", nombreSalida );
   else
   {
      printf( "Error: fichero: %s NO CERRADO\n", nombreSalida );
      return 1;
   }
   if( !fclose(ficheroEntrada) )
```

```
      printf( "Fichero: %s cerrado\n", nombreEntrada );
   else
   {
      printf( "Error: fichero: %s NO CERRADO\n", nombreEntrada );
      return 1;
   }

   return 0;
}
```

# Función ferror ANSI C

**Librería: stdio**

```
int ferror(FILE *stream);
```

La función *ferror* comprueba el indicador de errores para el stream apuntado por **stream**.

## Valor de retorno:

La función *ferror* retorna un valor distinto a cero si y sólo si el indicador de errores está activado para **stream**.

## Ejemplo:

```
#include <stdio.h>

int main( void )
{
   char c='z';
   char nombre[11] = "datos2.dat";
   FILE *fichero;

   if( NULL == (fichero = fopen(nombre, "r")) )
      printf( "ERROR: No se pudo abrir el fichero, %s\n", nombre );

   if( EOF == fputc( c, fichero ) )    printf( "fputc => Error\n" );    /* Creamos un
error */
   else   printf( "Caracter: %c, se ha escrito en el fichero: %s\n", c, nombre );
/* No se dará el caso */

   if( ferror(fichero) )
   {
      printf( "Se han encontrado errores al realizar una operacion con el fichero,
%s.\n", nombre );
      clearerr( fichero );
      printf( "Los errores han sido despejado para el stream.\n" );
   }
   else    printf( "No hubo errores con el fichero: %s\n", nombre );    /* Tampoco se
dará el caso */

   fclose( fichero );

   return 0;
}
```

# Función fflush ANSI C

**Librería: stdio**

```
int fflush(FILE *stream);
```

Si **stream** apunta a un stream de salida o de actualización cuya operación más reciente no era de entrada, la función *fflush* envía cualquier dato aún sin escribir al entorno local o a ser escrito en el fichero; si no, entonces el comportamiento no está definido. Si **stream** es un puntero nulo, la función *fflush* realiza el despeje para todos los streams cuyo comportamiento está descrito anteriormente.

## Valor de retorno:

La función *fflush* retorna cero si el stream fue despejado con éxito. Si se detectaron errores, entonces retorna EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
  char acumulador[BUFSIZ];

  setbuf( stdout, acumulador );

  printf( "Esto es una prueba\n" );
  printf( "Este mensaje se mostrara a la vez\n" );
  printf( "setbuf, acumula los datos en un puntero\n" );
  printf( "hasta que se llene completamente\n" );

  fflush( stdout );

  return 0;
}
```

# Función fgetc ANSI C

**Librería: stdio**

```
int fgetc(FILE *stream);
```

Esta función obtiene el carácter siguiente (si está presente) como un **unsigned char** convertido a **int**, desde el stream de entrada apuntado por **stream**, y avanza el indicador de posición de ficheros asociado al stream (si está definido).

## Valor de retorno:

La función *fgetc* retorna el carácter siguiente desde el stream de entrada apuntado por **stream**. Si el stream está en el final de fichero, el indicador del final de fichero para el stream es activado y *fgetc* retorna EOF. Si ocurre un error de lectura, el indicador de error para el stream es activado y *fgetc* retorna EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[10]="datos.dat";
   FILE *fichero;
   int i;

   fichero = fopen( nombre, "r" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "existe (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   printf( "Los 18 primeros caracteres del fichero: %s\n\n", nombre );
   for( i=1; i<=18; i++)   printf( "%c", fgetc(fichero) );

   if( !fclose(fichero) )
      printf( "\nFichero cerrado\n" );
   else
   {
      printf( "\nError: fichero NO CERRADO\n" );
      return 1;
   }
```

```
    return 0;
}
```

# Función fgetpos ANSI C

**Librería: stdio**

```
int fgetpos(FILE *stream, fpos_t *posicion);
```

La función *fgetpos* guarda el valor actual del indicador de posición de ficheros para el stream apuntado por **stream** en el objeto apuntado por **posicion**. El valor guardado contiene información no especificado servible a la función fsetpos para recolocar el stream a su posición cuando se llamó a la función *fgetpos*.

## Valor de retorno:

La función *fgetpos* retorna cero si es llevada a cabo con éxito. Si falla, la función retorna un valor distinto a cero y guarda un valor positivo, según la definición de la implementación, en errno.

## Ejemplo:

```
#include <stdio.h>

int main()
{
  char nombre[11] = "datos4.dat",
       mensaje[81]="Esto es nua rpueba usando fgetpos y fsetpos.";
  FILE *fichero;
  fpos_t posicion=0, comienzo;

  fichero = fopen( nombre, "w+" );
  printf( "Fichero: %s -> ", nombre );
  if( fichero )
    printf( "creado (ABIERTO)\n" );
  else
  {
    printf( "Error (NO ABIERTO)\n" );
    return 1;
  }

  fgetpos( fichero, &comienzo );
  printf( "Posicion del fichero: %d\n", posicion );

  fprintf( fichero, mensaje );
  printf( "\nEscrito: \"%s\"\n", mensaje );

  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  fsetpos( fichero, &comienzo );
```

```
  fprintf( fichero, "%s", "Esto es una prueba" );
  printf( "Corregiendo errores...Escrito: \"Esto es una prueba\"\n" );

  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  rewind( fichero );
  printf( "\"Rebobinando\" el fichero -> Vuelta al comienzo\n" );
  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  printf( "\nLeyendo del fichero \"%s\"\n", nombre );
  fgets( mensaje, 81, fichero );
  printf( "\"%s\"\n\n", mensaje );

  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  if( !fclose(fichero) )
    printf( "Fichero cerrado\n" );
  else
  {
    printf( "Error: fichero NO CERRADO\n" );
    return 1;
  }

  return 0;
}
```

# Función fgets ANSI C

**Librería: stdio**

```
char *fgets(char *cadena, int n, FILE *stream);
```

Esta función lee como máximo uno menos que el número de caracteres indicado por **n** desde el stream apuntado por **stream** al array apuntado por **cadena**. Ningún carácter adicional es leído después del carácter de nueva línea (el cual es retenido) o después de un final de fichero (EOF). Un carácter nulo es escrito inmediatamente después del último carácter leído en el array.

## Valor de retorno:

La función *fgets* retorna **cadena** si es realizada con éxito. Si un final de fichero (EOF) es encontrado y ningún carácter ha sido leído en el array, entonces el contenido del array permanece invariable y un puntero nulo es retornado. Si ocurre un error de lectura durante el proceso, el contenido del array es indeterminado y un puntero nulo es retornado.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[10]="datos.dat", linea[81];
   FILE *fichero;

   fichero = fopen( nombre, "r" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "existe (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   printf( "La primera linea del fichero: %s\n\n", nombre );
   printf( "%s\n", fgets(linea, 81, fichero) );
```

```
   if( !fclose(fichero) )
      printf( "\nFichero cerrado\n" );
   else
   {
      printf( "\nError: fichero NO CERRADO\n" );
      return 1;
   }

   return 0;
}
```

# Función floor ANSI C

**Librería: math**

```
double floor(double x);
```

Calcula el valor integral más grande que no sea mayor de **x**.

## Valor de retorno:

La función floor retorna el resultado de la función "suelo" de **x**.

## Ejemplo:

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

int main()
{
   double x = 6.54321;

   printf( "floor( %f ) = %f\n", x, floor(x) );
   return 0;
}
```

# Función fmod ANSI C

**Librería: math**

```
double fmod(double x, double y);
```

Calcula el resto de coma flotante de la división de **x**/**y**.

## Valor de retorno:

La función fmod retorna el resto de **x**/**y**.

## Ejemplo:

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

int main()
{
   double x = -6.54321, y = 1.23456;

   printf( "fmod( %f ) = %f\n", x, fmod(x) );
   return 0;
}
```

# Función fopen ANSI C

**Librería: stdio**

```
FILE *fopen(const char *nombre, const char *modo);
```

Abre un fichero cuyo nombre es la cadena apuntada por **nombre**, y adjudica un stream a ello. El argumento **modo** apunta a una cadena empezando con una serie de caracteres de la siguiente secuencia:

| | |
|---|---|
| **r** | Abre un fichero de texto para lectura |
| **w** | Trunca, a longitud cero o crea un fichero de texto para escribir |
| **a** | Añade; abre o crea un fichero de texto para escribir al final del fichero (EOF) |
| **rb** | Abre un fichero en modo binario para lectura |
| **wb** | Trunca, a longitud cero o crea un fichero en modo binario para escribir |
| **ab** | Añade; abre o crea un fichero en modo binario para escribir al final del fichero (EOF) |
| **r+** | Abre un fichero de texto para actualización (lectura y escritura) |
| **w+** | Trunca, a longitud cero o crea un fichero de texto para actualización |
| **a+** | Añade; abre o crea un fichero de texto para actualización, escribiendo al final del fichero (EOF) |
| **r+b** ó **rb+** | Abre un fichero en modo binario para actualización (lectura y escritura) |
| **w+b** ó **wb+** | Trunca, a longitud cero o crea un fichero en modo binario para actualización |
| **a+b** ó **ab+** | Añade; abre o crea un fichero en modo binario para actualización, escribiendo al final del fichero (EOF) |

Abriendo un fichero con el modo de lectura ('r' como el primer carácter del argumento **modo**) fallará si el fichero no existe o no puede ser leído. Abriendo el fichero con el modo de añadidura ('a' como el primer carácter del argumento **modo**) ocasiona todas las escrituras posteriores al fichero a ser forzadas al final de fichero (EOF) actual, sin considerar llamadas interventivas a la función fseek. En algunas implementaciones,

abriendo un fichero en modo binario con el modo de añadidura ('b'como el segundo o tercer carácter del argumento **modo**) puede colocar, inicialmente, el indicador de posición de ficheros para el stream más allá de los últimos datos escritos, debido al relleno de caracteres nulos.

Cuando un fichero es abierto con el modo de actualización ('+' como el segundo o tercer carácter del argumento **modo**), la entrada y salida pueden ser manipuladas en el stream asociado. Sin embargo, la salida no puede estar seguida directamente de una entrada sin tener una llamada interventiva a la función fflush o a una función de manipulación del fichero de posición (fseek, fsetpos, o rewind), y la entrada no puede estar seguida directamente de una salida sin tener una llamada interventiva a una función de manipulación del fichero de posición, al menos que el proceso de entrada encuentre un final de fichero (EOF). Abriendo (o creando) un fichero de texto con el modo de actualización puede en su lugar abrir (o crear) un stream binario en algunas implementaciones.

Cuando es abierto, un stream es almacenado completamente si, y sólo si, puede ser determinado que no hace referencia a un dispositivo interactivo. Los indicadores de error y final de fichero (EOF) para el stream son borrados.

## Valor de retorno:

La función *fopen* retorna un puntero al objeto controlando el stream. Si el proceso de abertura no es realizado acabo, entonces retorna un puntero nulo.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   FILE *fichero;
   char nombre[10] = "datos.dat";

   fichero = fopen( nombre, "w" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }
```

```
   if( !fclose(fichero) )
      printf( "Fichero cerrado\n" );
   else
   {
      printf( "Error: fichero NO CERRADO\n" );
      return 1;
   }

   return 0;
}
```

# Especificadores de Conversión para fscanf ANSI C

**Librería: stdio**

Enteros

| | |
|---|---|
| **d** | Lee un entero decimal opcionalmente con signo. El argumento correspondiente es un puntero a un entero |
| **i** | Lee un entero decimal, octal, o hexadecimal opcionalmente con signo. El argumento correspondiente es un puntero a un entero |
| **o** | Lee un entero octal sin signo. El argumento correspondiente es un puntero a un entero sin signo |
| **u** | Lee un entero decimal sin signo. El argumento correspondiente es un puntero a un entero sin signo |
| **x** ó **X** | Lee un entero hexadecimal. El argumento correspondiente es un puntero a un entero sin signo. |
| **h** ó **l** | [Prefijo] Coloca antes de cualquier especificador de conversión de enteros para indicar que la entrada de datos es un entero de tipo **short** o **long** |

## Ejemplo:

```
#include <stdio.h>

int main()
{
   int ddec, idec, ioct, ihex;
   unsigned int ooct, udec, xhex;
   short int hddec;
   long int lddec;

   printf( "Introduce un número decimal: " );
   scanf( "%d", &ddec );

   printf( "Introduce un número decimal, octal (precédelo con un cero, 0), y\n
hexadecimal (precédelo con un cero y una '\x\', 0x): " );
   scanf( "%i %i %i", &idec, &ioct, &ihex );

   printf( "Introduce un número octal: " );
   scanf( "%o", &ooct );

   printf( "Introduce un número decimal (no negativo): " );
   scanf( "%u", &udec );

   printf( "Introduce un número hexadecimal: " );
   scanf( "%x", &xhex );

   printf( "Introduce un número decimal \"pequeño\" y uno \"grande\": " );
   scanf( "%hd %ld", &hddec, &lddec );

   printf( "\nHas introducido: " );
   printf( "%d %d %d %d %d %d %d %d %d\n", ddec, idec, ioct, ihex, ooct, udec, xhex,
hddec, lddec );

   return 0;
}
```

Números de Coma Flotante

| | |
|---|---|
| **e**, **E**, **f**, **g**, ó **G** | Lee un valor de coma flotante. El argumento correspondiente es un puntero a una variable de coma flotante |
| **l** ó **L** | [Prefijo] Coloca antes de cualquier especificador de conversión de coma flotante para indicar que la entrada de datos es un valor de tipo **double** o **long double** |

## Ejemplo:

```
#include <stdio.h>

int main()
{
   float ereal, freal, greal;
   double lfreal;
   long double Lfreal;

   printf( "Introduce un número real: " );
   scanf( "%f", &freal );

   printf( "Introduce un número real, con exponente: " );
   scanf( "%e", &ereal );

   printf( "Introduce otro número real, con exponente: " );
   scanf( "%g", &greal );

   printf( "Introduce un número real con doble precisión y otro con mayor aún: " );
   scanf( "%lf %Lf", &lfreal, &Lfreal );

   printf( "\nHas introducido: " );
   printf( "%5.10f %5.12e %5.12g %10.20lf %10.20Lf\n", freal, ereal, greal, lfreal,
Lfreal );

   return 0;
}
```

Caracteres y Cadenas

| | |
|---|---|
| **c** | Lee un carácter. El argumento correspondiente es un puntero a **char**, ningún nulo ('**\0**') es añadido |
| **s** | Lee una cadena de caracteres. El argumento correspondiente es un puntero a una array de tipo **char** que es suficientemente grande para guardar la cadena y un carácter nulo ('**\0**') final |

## Ejemplo:

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

int main()
{
   char c, cadena[80];

   printf( "Introduce un carácter: " );
   scanf( "%c", &c );

   printf( "Introduce una cadena de caracteres (sin espacios): " );
   scanf( "%s", cadena );
```

```
   printf( "\nHas introducido: " );
   printf( "\'%c\' \"%s\" (%d caracteres)\n", c, cadena, strlen(cadena) );

   return 0;
}
```

Conjunto de Búsqueda

| | |
|---|---|
| **[***caracteres a buscar***]** | Busca en una cadena de caracteres un conjunto de caracteres que están guardados en un array |

## Ejemplo:

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

int main()
{
   char cadena[80], temp[80];

   printf( "Escribe una palabra, que empiece por una vocal: " );
   scanf( "%[aeiou]", cadena );

   scanf( "%s", temp );    /* Recoger los caracteres restantes */

   printf( "\nSólo se ha recogido las primeras letras que sean vocales: \"%s\"\n",
cadena );

   printf( "Escribe otra palabra, que empiece por una consonante (no son vocales): "
);
   scanf( "%[^aeiou]", cadena );

   printf( "\nSólo se ha recogido las primeras letras que no sean vocales: \"%s\"\n",
cadena );

   return 0;
}
```

Misceláneo

| | |
|---|---|
| *números enteros* | [Prefijo] Coloca antes de un especificador para indicar la "anchura de campo" de caracteres, a la hora de asignar el valor introducido. Sólo se asignarán los primeros **x** caracteres especificados por el número prefijo. |
| **p** | Lee un puntero de memoria de la misma forma producida cuando una dirección de memoria es convertida con **%p** en una sentencia de fprintf, printf, sprintf, vfprintf, vprintf, o vsprintf |
| **n** | Guarda el número de caracteres entradas hasta ahora en este fscanf, scanf, o sscanf. El argumento correspondiente es un puntero a un entero. |
| **%** | Salta un signo de porcentaje (**%**) en los datos de entrada |
| ***** | [Prefijo] Símbolo de supresión de asignación. Actúa como un comodín para el dato introducido. |

## Ejemplo:

```
#include <stdio.h>

int main()
{
   int num1, num2, tam, porc;
   unsigned char cadena[10];
   unsigned int dia, mes, anyo;
   void *memPtr;

   printf( "Introduce un número de 6 dígitos: " );
   scanf( "%3d%3d", &num1, &num2 );
   printf( "El número introducido ha sido separado en 2 números de 3 dígitos cada
uno: %d y %d\n\n", num1, num2 );

   printf( "Introduce una palabra (hasta 10 letras): " );
   scanf( "%s%n", cadena, &tam );
   printf( "La palabra escrita: \"%s\", %d caracteres escritos. Comienza en la
dirección: %p\n\n", cadena, tam, cadena );

   printf( "Introduce una dirección de memoria, en hexadecimal: " );
   scanf( "%p", &memPtr );
   printf( "La memoria es: %p, y como datos tiene: %s\n\n", memPtr, memPtr );

   printf( "Introduce un porcentaje (usa el símbolo %%): " );
   scanf( "%d%%", &porc );
   printf( "Introdujiste: %d%%\n\n", porc );

   printf( "Introduce la fecha de hoy: " );
   scanf( "%d%*c%d%*c%d", &dia, &mes, &anyo );
   printf( "Día: %d, Mes: %d, Año: %d\n\n", dia, mes, anyo );

   return 0;
}
```

# Función fprintf ANSI C

**Librería: stdio**

```
int fprintf(FILE *stream, const char *formato, ...);
```

Esta función envía datos al stream apuntado por **stream**, bajo el control de la cadena apuntada por **formato** que especifica cómo los argumentos posteriores son convertidos para la salida. Si hay argumentos insuficientes para el formato, el comportamiento no está definido. Si el formato termina mientras quedan argumentos, los argumentos restantes son evaluados (como siempre) pero ignorados. La función retorna control cuando el final de la cadena de formato es encontrado.

## Valor de retorno:

La función *fprintf* retorna el número de caracteres transmitidos, o un valor negativo si un error de salida se ha producido.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   FILE *fichero;
   char nombre[10] = "datos.dat";
   unsigned int i;

   fichero = fopen( nombre, "w" );
   printf( "Fichero: %s (para escritura) -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   fprintf( fichero, "Esto es un ejemplo de usar la funcion \'fprintf\'\n" );
   fprintf( fichero, "\t 2\t 3\t 4\n" );
   fprintf( fichero, "x\tx\tx\tx\n\n" );
   for( i=1; i<=10; i++ )
      fprintf( fichero, "%d\t%d\t%d\t%d\n", i, i*i, i*i*i, i*i*i*i );

   fprintf( stdout, "Datos guardados en el fichero: %s\n", nombre );
   if( !fclose(fichero) )
      printf( "Fichero cerrado\n" );
   else
   {
      printf( "Error: fichero NO CERRADO\n" );
      return 1;
   }
```

```
    return 0;
}
```

# Función fputc ANSI C

**Librería: stdio**

```
int fputc(int c, FILE *stream);
```

Esta función escribe el carácter indicado por **c** (convertido a un **unsigned char**) al stream de salida apuntado por **stream**, en la posición indicada por el indicador de posición de ficheros asociado al stream (si está definido), y avanza el indicador apropiadamente. Si el fichero no soporta peticiones de posición, o si el stream fue abierto con el modo de añado, el carácter es añadido al stream de salida.

## Valor de retorno:

La función *fputc* retorna el carácter escrito. Si ocurre un error de escritura, el indicador de error para el stream es activado y *fputc* retorna EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[10]="datos.dat";
   FILE *fichero;
   int i;

   fichero = fopen( nombre, "a" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "existe o ha sido creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   printf( "Escribimos las 18 primeras letras del abecedario ingles en el fichero:
%s\n\n", nombre );
   for( i=0; i<18; i++)   printf( "%c", fputc('a'+i, fichero) );

   if( !fclose(fichero) )
      printf( "\nFichero cerrado\n" );
   else
   {
      printf( "\nError: fichero NO CERRADO\n" );
      return 1;
   }

   return 0;
}
```

# Función fputs ANSI C

**Librería: stdio**

```
int fputs(const char *cadena, FILE *stream);
```

Esta función escribe la cadena apuntada por **cadena** al stream apuntado por **stream**. El carácter nulo no es escrito.

## Valor de retorno:

La función *fputs* retorna EOF si ocurre un error de escritura, si no, entonces retorna un valor no negativo.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[11]="datos2.dat";
   FILE *fichero;

   fichero = fopen( nombre, "w" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   fputs( "Esto es un ejemplo usando \'fputs\'\n", fichero );

   if( !fclose(fichero) )
      printf( "\nFichero cerrado\n" );
   else
   {
      printf( "\nError: fichero NO CERRADO\n" );
      return 1;
   }
```

```
    return 0;
}
```

# Función fread ANSI C

**Librería: stdio**

```
size_t fread(void *puntero, size_t tamanyo, size_t nmemb, FILE *stream);
```

La función *fread* recibe, en el array apuntado por **puntero**, hasta **nmemb** de elementos cuyo tamaño es especificado por **tamanyo**, desde el stream apuntado por **stream**. El indicador de posición de ficheros para el stream (si está definido) es avanzado por el número de caracteres leídos correctamente. Si existe un error, el valor resultante del indicador de posición de ficheros para el stream es indeterminado. Si un elemento es parcialmente leído, entonces su valor es indeterminado.

## Valor de retorno:

La función *fread* retorna el número de caracteres leídos correctamente, el cual puede ser menor que **nmemb** si se encuentra un error de lectura o un final de fichero. Si **tamanyo** o **nmemb** es cero, *fread* retorna cero, y el contenido del array y el estado del stream permanecen invariados.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   FILE *fichero;
   char nombre[11] = "datos5.dat";
   unsigned int dinero[10] = { 23, 12, 45, 345, 512, 345, 654, 287, 567, 124 };
   unsigned int leer[10], i;

   fichero = fopen( nombre, "w+" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   printf( "Escribiendo cantidades:\n\n" );

   for( i=0; i<10; i++ )
     printf( "%d\t", dinero[i] );

   fwrite( dinero, sizeof(unsigned int), 10, fichero );

   printf( "\nLeyendo los datos del fichero \"%s\":\n", nombre );
   rewind( fichero );
   fread( leer, sizeof(unsigned int), 10, fichero );

   for( i=0; i<10; i++ )
     printf( "%d\t", leer[i] );

   if( !fclose(fichero) )
```

```
      printf( "\nFichero cerrado\n" );
   else
   {
      printf( "\nError: fichero NO CERRADO\n" );
      return 1;
   }

   return 0;
}
```

# Función free ANSI C

**Librería: stdlib**

```
void free(void *ptr);
```

Causa el espacio apuntado por **ptr** a ser desadjudicado, esto es, ser disponible para otra adjudicación. Si **ptr** es un puntero nulo, no se realizará ninguna acción. De lo contrario, si el argumento no corresponde a un puntero previamente retornado por la función calloc, malloc, o realloc, o si el espacio ha sido desadjudicado por una llamada a *free* o realloc, el comportamiento no está definido.

## Valor de retorno:

La función *free* no retorna ningún valor.

## Ejemplo:

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

int main( void )
{
   int *numPtr, i;
   size_t tamanyo=0;

   printf( "Introduzca el tamaño de la lista: " );
   scanf( "%d", &tamanyo );

   puts( "Adjudicamos espacio a la lista." );
   numPtr = (int *)malloc( tamanyo*sizeof(int) );

   for( i=0; i<tamanyo-1; i++ )
      printf( "%d, ", numPtr[i] = rand() % 100 + 1 );
   printf( "%d\n", numPtr[i] = rand() % 100 + 1 );

   numPtr = (int *)realloc( numPtr, tamanyo/=2 );
   printf( "Recortamos la lista a la mitad: %d\n", tamanyo );
   for( i=0; i<tamanyo-1; i++ )
      printf( "%d, ", numPtr[i] );
   printf( "%d\n", numPtr[i] );
```

```
    puts( "Liberamos el espacio." );
    free( numPtr );

    return 0;
}
```

# Función freopen ANSI C

**Librería: stdio**

```
FILE *freopen(const char *nombre, const char *modo, FILE *stream);
```

Abre un fichero cuyo nombre es la cadena apuntada por **nombre** y adjudica un stream a ello apuntado por **stream**. El argumento **modo** es usado tal como en la función fopen.

La función *freopen* primeramente intenta cerrar cualquier fichero que es asociado con el stream especificado. El error de no cerrar el fichero con éxito es ignorado. Los indicadores de error y final de fichero EOF) para el stream son borrados.

## Valor de retorno:

La función *freopen* retorna un puntero nulo si el proceso de abertura falla. De no ser así, *freopen* retorna el valor de **stream**.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   FILE *fichero;
   char nombre[10] = "datos.dat";

   fichero = fopen( nombre, "w" );
   printf( "Fichero: %s (para escritura) -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   printf( "Fichero: %s (para lectura) -> ", nombre );
   if( freopen(nombre, "r", fichero) == NULL )
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }
   else   printf( "listo para leer (ABIERTO)\n" );
```

```
   if( !fclose(fichero) )
      printf( "Fichero cerrado\n" );
   else
   {
      printf( "Error: fichero NO CERRADO\n" );
      return 1;
   }

   return 0;
}
```

# Función frexp ANSI C

**Librería: math**

```
double frexp(double valor, int *exp);
```

Parte en dos el número de coma flotante en una fracción normalizada y un entero con potencia a la 2. Guarda el entero en el objecto int apuntado por **exp**.

## Valor de retorno:

La función frexp retorna el valor de **x** tal que **x** es un double con magnitud en el intervalo [1/2, 1] o cero, y **valor** = **x** * 2$^{*\mathbf{exp}}$. Si **valor** es cero, ambas partes del resultado son cero.

## Ejemplo:

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

int main()
{
   double valor = 1.00, resultado;
   int *exp = 2;

   resultado = frexp( valor, exp );
   printf( "frexp( %f, %d ) = %f\n", valor, *exp, resultado );
   return 0;
}
```

# Función fscanf ANSI C

**Librería: stdio**

```
int fscanf(FILE *stream, const char *formato, ...);
```

Esta función recibe datos del stream apuntado por **stream**, bajo el control de la cadena apuntada por **formato** que especifica las secuencias de entrada permitadas y cómo han de ser convertidas para la asignación. Si hay argumentos insuficientes para el formato, el comportamiento no está definido. Si el formato termina mientras quedan argumentos, los argumentos restantes son evaluados (como siempre) pero ignorados. La función retorna control cuando el final de la cadena de formato es encontrado.

## Valor de retorno:

La función *fscanf* retorna el número de datos de entrada asignados, que puede ser menor que ofrecido, incluso cero, en el caso de un error de asignación. Si un error de entrada ocurre antes de cualquier conversión, la función *fscanf* retorna el valor de la macro EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   FILE *fichero;
   char nombre[10] = "datos.dat";
   unsigned int i, x1, x2, x3, x4;

   fichero = fopen( nombre, "r" );
   printf( "Fichero: %s (para lectura) -> ", nombre );
   if( fichero )
      printf( "existe (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   printf( "Datos leidos del fichero: %s\n", nombre );
   printf( "Esto es un ejemplo de usar la funcion \'fprintf\'\n" );
   printf( "\t 2\t 3\t 4\n" );
   printf( "x\tx\tx\tx\n\n" );

   fscanf( fichero, "Esto es un ejemplo de usar la funcion \'fprintf\'\n" );
   fscanf( fichero, "\t 2\t 3\t 4\n" );
   fscanf( fichero, "x\tx\tx\tx\n\n" );
   for( i=1; i<=10; i++ )
   {
      fscanf( fichero, "%d\t%d\t%d\t%d\n", &x1, &x2, &x3, &x4 );
      printf( "%d\t%d\t%d\t%d\n", x1, x2, x3, x4 );
   }
```

```
   if( !fclose(fichero) )
      printf( "Fichero cerrado\n" );
   else
   {
      printf( "Error: fichero NO CERRADO\n" );
      return 1;
   }

   return 0;
}
```

# Función fseek ANSI C

**Librería: stdio**

```
int fseek(FILE *stream, long int desplazamiento, int origen);
```

La función *fseek* activa el indicador de posición de ficheros para el stream apuntado por **stream**. Para un stream binario, la nueva posición, medido en caracteres del principio del fichero, es obtenida mediante la suma de **desplazamiento** y la posición especificada por **origen**. La posición especificada es el comienzo del fichero si **origen** es SEEK_SET, el valor actual del indicador de posición de fichero si es SEEK_CUR, o final de fichero si es SEEK_END. Un stream binario realmente no necesita soportar llamadas a *fseek* con un valor de **origen** de SEEK_END. Para un stream de texto, o bien **desplazamiento** será cero, o bien **desplazamiento** será un valor retornado por una llamada anterior a la función ftell al mismo stream y **origen** será SEEK_SET. Una llamada correcta a la función *fseek* despeja el indicador de final de fichero para el stream y deshace cualquier efecto producido por la función ungetc en el mismo stream. Después de una llamada a *fseek*, la siguiente operación en un stream de actualización puede ser de entrada o salida.

## Valor de retorno:

La función *fseek* retorna un valor distinto a cero sólo si una petición no se pudo satisfacer.

## Ejemplo:

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

int main()
{
  char nombre[11] = "datos4.dat", mensaje[81]="";
  FILE *fichero;
  long int comienzo, final;

  fichero = fopen( nombre, "r" );
  printf( "Fichero: %s -> ", nombre );
  if( fichero )
    printf( "existe (ABIERTO)\n" );
  else
  {
    printf( "Error (NO ABIERTO)\n" );
    return 1;
  }

  if( (comienzo=ftell( fichero )) < 0 )   printf( "ERROR: ftell no ha funcionado\n"
);
  else    printf( "Posición del fichero: %d\n\n", comienzo );

  fseek( fichero, 0L, SEEK_END );
  final = ftell( fichero );

  fseek( fichero, 0L, SEEK_SET );
  fgets( mensaje, 80, fichero );
  printf( "Tamaño del fichero \"%s\": %d bytes\n", nombre, final-comienzo+1 );
  printf( "Mensaje del fichero:\n%s\n", mensaje );
  printf( "\nTamaño del mensaje (usando strlen): %d\n", strlen(mensaje) );

  if( !fclose(fichero) )
```

```
    printf( "Fichero cerrado\n" );
  else
  {
    printf( "Error: fichero NO CERRADO\n" );
    return 1;
  }

  return 0;
}
```

# Función fsetpos ANSI C

**Librería: stdio**

```
int fsetpos(FILE *stream, const fpos_t *posicion);
```

La función *fsetpos* activa el indicador de posición de ficheros para el stream apuntado por **stream** según el valor del objeto apuntado por **posicion**, el cual será un valor obtenido de una llamada previa a la función fgetpos del mismo stream. Una llamada correcta a la función *fsetpos* despeja el indicador de final de fichero para el stream y deshace cualquier efecto producido por la función ungetc en el mismo stream. Después de una llamada a *fsetpos*, la siguiente operación en un stream de actualización puede ser de entrada o salida.

## Valor de retorno:

La función *fsetpos* retorna cero si es llevada a cabo con éxito. Si falla, la función retorna un valor distinto a cero y guarda un valor positivo, según la definición de la implementación, en errno.

## Ejemplo:

```
#include <stdio.h>

int main()
{
  char nombre[11] = "datos4.dat",
       mensaje[81]="Esto es nua rpueba usando fgetpos y fsetpos.";
  FILE *fichero;
  fpos_t posicion=0, comienzo;

  fichero = fopen( nombre, "w+" );
  printf( "Fichero: %s -> ", nombre );
  if( fichero )
    printf( "creado (ABIERTO)\n" );
  else
  {
    printf( "Error (NO ABIERTO)\n" );
    return 1;
  }

  fgetpos( fichero, &comienzo );
  printf( "Posicion del fichero: %d\n", posicion );

  fprintf( fichero, mensaje );
  printf( "\nEscrito: \"%s\"\n", mensaje );

  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );
```

```
  fsetpos( fichero, &comienzo );
  fprintf( fichero, "%s", "Esto es una prueba" );
  printf( "Corregiendo errores...Escrito: \"Esto es una prueba\"\n" );

  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  rewind( fichero );
  printf( "\"Rebobinando\" el fichero -> Vuelta al comienzo\n" );
  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  printf( "\nLeyendo del fichero \"%s\"\n", nombre );
  fgets( mensaje, 81, fichero );
  printf( "\"%s\"\n\n", mensaje );

  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  if( !fclose(fichero) )
    printf( "Fichero cerrado\n" );
  else
  {
    printf( "Error: fichero NO CERRADO\n" );
    return 1;
  }

  return 0;
}
```

# Función ftell ANSI C

**Librería: stdio**

```
long int ftell(FILE *stream);
```

La función *fseek* obtiene el valor actual del indicador de posición de fichero para el stream apuntado por **stream**. Para un stream binario, el valor es el número de caracteres desde el principio del fichero. Para un stream de texto, su indicador de posición de fichero contiene información no especificado, servible a la función fseek para retornar el indicador de posición de fichero para el stream a su posición cuando se llamó a *ftell*; la diferencia entre los dos valores de retorno no es necesariamente una medida real del número de caracteres escritos o leídos.

## Valor de retorno:

La función *ftell* retorna el valor del indicador de posición de fichero para el stream, si se tiene éxito. Si falla, la función *ftell* retorna **-1L** y guarda un valor positivo, según la definición de la implementación, en errno.

## Ejemplo:

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

int main()
{
  char nombre[11] = "datos4.dat", mensaje[81]="";
  FILE *fichero;
  long int comienzo, final;

  fichero = fopen( nombre, "r" );
  printf( "Fichero: %s -> ", nombre );
  if( fichero )
    printf( "existe (ABIERTO)\n" );
  else
  {
    printf( "Error (NO ABIERTO)\n" );
    return 1;
  }

  if( (comienzo=ftell( fichero )) < 0 )   printf( "ERROR: ftell no ha funcionado\n"
);
  else    printf( "Posicion del fichero: %d\n\n", comienzo );

  fseek( fichero, 0L, SEEK_END );
  final = ftell( fichero );

  fseek( fichero, 0L, SEEK_SET );
  fgets( mensaje, 80, fichero );
  printf( "Tama¤o del fichero \"%s\": %d bytes\n", nombre, final-comienzo+1 );
  printf( "Mensaje del fichero:\n%s\n", mensaje );
  printf( "\nTama¤o del mensaje (usando strlen): %d\n", strlen(mensaje) );

  if( !fclose(fichero) )
    printf( "Fichero cerrado\n" );
  else
  {
    printf( "Error: fichero NO CERRADO\n" );
```

```
    return 1;
  }

  return 0;
}
```

# Función fwrite ANSI C

**Librería: stdio**

```
size_t fwrite(const void *puntero, size_t tamanyo, size_t nmemb, FILE *stream);
```

La función *fwrite* envía, desde el array apuntado por **puntero**, hasta **nmemb** de elementos cuyo tamaño es especificado por **tamanyo**, al stream apuntado por **stream**. El indicador de posición de ficheros para el stream (si está definido) es avanzado por el número de caracteres escritos correctamente. Si existe un error, el valor resultante del indicador de posición de ficheros para el stream es indeterminado.

## Valor de retorno:

La función *fwrite* retorna el número de caracteres escritos correctamente, el cual puede ser menor que **nmemb**, pero sólo si se produce un error de escritura.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   FILE *fichero;
   char nombre[11] = "datos5.dat";
   unsigned int dinero[10] = ;
   unsigned int leer[10], i;

   fichero = fopen( nombre, "w+" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   printf( "Escribiendo cantidades:\n\n" );

   for( i=0; i<10; i++ )
     printf( "%d\t", dinero[i] );

   fwrite( dinero, sizeof(unsigned int), 10, fichero );

   printf( "\nLeyendo los datos del fichero \"%s\":\n", nombre );
   rewind( fichero );
   fread( leer, sizeof(unsigned int), 10, fichero );

   for( i=0; i<10; i++ )
     printf( "%d\t", leer[i] );

   if( !fclose(fichero) )
      printf( "\nFichero cerrado\n" );
   else
```

```
   {
      printf( "\nError: fichero NO CERRADO\n" );
      return 1;
   }

   return 0;
}
```

# Función getc ANSI C

**Librería: stdio**

```
int getc(FILE *stream);
```

Esta función es equivalente a fgetc, excepto que si es implementado como una macro, puede evaluar **stream** más de una vez - el argumento debería ser una expresión sin efectos secundarios.

## Valor de retorno:

La función *getc* retorna el carácter siguiente desde el stream de entrada apuntado por **stream**. Si el stream está en el final de fichero, el indicador del final de fichero para el stream es activado y *getc* retorna EOF. Si ocurre un error de lectura, el indicador de error para el stream es activado y *getc* retorna EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[10]="datos.dat";
   FILE *fichero;
   int i;

   fichero = fopen( nombre, "r" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "existe (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   printf( "Los 18 primeros caracteres del fichero: %s\n\n", nombre );
   for( i=1; i<=18; i++)   printf( "%c", getc(fichero) );

   if( !fclose(fichero) )
      printf( "\nFichero cerrado\n" );
   else
   {
      printf( "\nError: fichero NO CERRADO\n" );
      return 1;
   }
```

```
    return 0;
}
```

# Función getchar ANSI C

**Librería: stdio**

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

Esta función es equivalente a getc con el argumento stdin.

## Valor de retorno:

La función *getchar* retorna el carácter siguiente desde el stream de entrada apuntado por **stream**. Si el stream está en el final de fichero, el indicador del final de fichero para el stream es activado y *getchar* retorna EOF. Si ocurre un error de lectura, el indicador de error para el stream es activado y *getchar* retorna EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[20]="";
   int i;

   printf( "Elige el numero en el menu:\n\n" );
   printf( "1 - Cargar fichero\n2 - Guardar en un fichero\n3 - Otras operaciones\n4 -
Mostrar datos\n0 - Salir\n\n );
   printf( "Opcion: " );
   printf( "\nHas elegido: %c", getchar() );

   return 0;
}
```

# Función getenv ANSI C

**Librería: stdlib**

```
char *getenv(const char *nombre);
```

Busca una "lista de entorno", proporcionado por el entorno local, para una cadena que empareje la cadena apuntada por **nombre**. El conjunto de nombres del entorno y el método para alterar la lista de entorno están definidos según la implementación.

## Valor de retorno:

La función *getenv* retorna un puntero a la cadena asociado con el miembro emparejado de la lista. La cadena que apunta a ello no será modificado por el programa, pero puede ser sobreescrito por una llamada posterior a la función *getenv*. Si el **nombre** especificado no puede ser encontrado, un puntero nulo es retornado.

## Ejemplo:

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

int main( void )
{
   char *directorioPtr, *pathPtr;

   pathPtr = getenv( "PATH" );
   puts( "La lista de directorios en el PATH es la siguiente:" );

   directorioPtr = strtok(pathPtr, ";\n" );
   puts( directorioPtr );
   while( (directorioPtr = strtok(NULL, ";\n")) != NULL )
      puts( directorioPtr );

   return 0;
}
```

# Función gets ANSI C

**Librería: stdio**

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

Esta función lee caracteres desde el stream apuntado por **stream** stdin, en el array apuntado por **cadena**, hasta que se encuentre un final de fichero (EOF) o un carácter de línea nueva es leído. Cualquier carácter de línea nueva es descartado, y un carácter nulo es escrito inmediatamente después del último carácter leído en el array.

## Valor de retorno:

La función *gets* retorna **cadena** si es realizada con éxito. Si un final de fichero (EOF) es encontrado y ningún carácter ha sido leído en el array, entonces el contenido del array permanece invariable y un puntero nulo es retornado. Si ocurre un error de lectura durante el proceso, el contenido del array es indeterminado y un puntero nulo es retornado.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char oracion[81];

   printf( "Escribe una oracion:\n");
   printf( "\nHas escrito: \"%s\"\n", gets(oracion) );

   return 0;
}
```

# Función gmtime ANSI C

**Librería: time**

```
struct tm *gmtime(const time_t *tiempoPtr);
```

La función *gmtime* convierte el tiempo en formato condensado apuntado por **tiempoPtr** en el tiempo en formato separado, expresado como Tiempo Universal Coordenada (UTC).

## Valor de retorno:

La función *gmtime* retorna un puntero al objeto, o un puntero nul si UTC no está disponible.

## Ejemplo:

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

int main( void )
{
   long int i=0;
   time_t comienzo, final;
   struct tm *tiempoComienzoPtr, *tiempoFinalPtr;

   comienzo = time( NULL );
   for( i=0; i<10000; i++ )   printf( "-" );
   final = time( NULL );

   printf( "Comienzo: %u s\n", comienzo );
   printf( "Final: %u s\n", final );
   printf( "Número de segundos transcurridos desde el comienzo del programa: %f s\n",
difftime(final, comienzo) );

   tiempoComienzoPtr = gmtime( &comienzo );
   tiempoFinalPtr = gmtime( &final );
   printf( "Comienzo: %s\n", asctime(tiempoComienzoPtr) );
   printf( "Final: %s\n", asctime(tiempoFinalPtr) );

   return 0;
}
```

# Función labs ANSI C

**Librería: stdlib**

```
long int labs(long int num);
```

Similar a abs, excepto que el argumento es de tipo **long int**.

## Valor de retorno:

La función *labs* retorna el valor absoluto (de tipo **long int**).

## Ejemplo:

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

int main()
{
   long int num;

   puts( "Escriba un numero entero:" );
   scanf( "%d", &num );
   printf( "labs( %d ) = %d\n", num, labs(num) );

   return 0;
}
```

# Función ldexp ANSI C

**Librería: math**

```
double ldexp(double x, int exp);
```

Multiplica un número de coma flotante y un entero con potencia a la 2. Puede producirse un error de recorrido.

## Valor de retorno:

La función ldexp retorna el valor de **x** multiplicado por 2 elevado a la potencia de **exp**, es decir, **x** * 2 **exp**.

## Ejemplo:

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

int main()
{
   double x = 1.00;
   int exp = 2;

   printf( "ldexp( %f, %d ) = %f\n", x, exp, ldexp(x,exp) );
   return 0;
}
```

# Función ldiv ANSI C

**Librería: stdlib**

```
ldiv_t ldiv(long int num, long int denom);
```

Similar a la función div, excepto que los argumentos son de tipo **long int**.

## Valor de retorno:

La función *ldiv* retorna la estructura de tipo ldiv_t, conteniendo el cociente y el resto (de tipo **long int**).

## Ejemplo:

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

int main()
{
   ldiv_t ld;
   long int num, denom;

   puts( "Escriba el numerador y el denominador (separados por un espacio):" );
   scanf( "%d %d", &num, &denom );
   ld = ldiv( num, denom );
   printf( "ldiv( %d, %d ) : cociente = %d, resto = %d\n", num, denom, ld.quot,
ld.rem );

   return 0;
}
```

# Función localeconv ANSI C

**Librería: locale**

```
struct lconv *localeconv(void );
```

La función *localeconv* asigna los componentes de un objeto con tipo struct lconv con valores apropiados para el formateo de cantidades numéricas (monetarias y otras) según las reglas de la localidad actual.

Los miembros de la estructura con tipo **char *** son punteros a cadenas, cualquiera de ellos (excepto **decimal_point**) puede apuntar a "", para indicar que el valor no está disponible en la localidad actual o es de longitud cero. Los miembros con tipo **char** no son números negativos, cualquiera de ellos puede ser CHAR_MAX para indicar que el valor no está disponible en la localidad actual. Los miembros incluyen los siguientes:

**char *decimal_point**

El carácter de la coma decimal usado para formatear cantidades no monetarias.

**char *thousands_sep**

El carácter usado para separa grupos de dígitos antes del carácter de la coma decimal en cantidades formateadas no monetarias.

**char *grouping**

Una cadena cuyos elementos indican el tamaño de cada grupo de dígitos en cantidades formateadas no monetarias.

**char *int_curr_symbol**

El símbolo internacional de la unidad monetaria aplicable a la localidad actual. Los tres primeros caracteres contienen el símbolo internacional de la unidad monetaria con acorde con aquéllos especificados en ISO 4217:1987. El cuarto carácter (inmediatamente precediendo el carácter nulo) es el carácter usado para separar el símbolo internacional de la unidad monetaria de la cantidad monetaria.

**char *currency_symbol**

El símbolo local de la unidad monetaria aplicable a la localidad actual.

**char *mon_decimal_point**

La coma decimal usada para formatear cantidades monetarias.

**char *mon_thousands_sep**

El separador de grupos de dígitos antes de la coma decimal en cantidades formateadas monetarias.

**char *mon_grouping**

Una cadena cuyos elementos indican el tamaño de cada grupo de dígitos en cantidades formateadas monetarias.

**char *positive_sign**

La cadena usada para indicar valores no negativos de cantidades formateadas monetarias.

**char *negative_sign**

La cadena usada para indicar valores negativos de cantidades formateadas monetarias.

**char int_frac_digits**

El número de dígitos fraccionarios (aquéllos después de la coma decimal) para ser mostrado en una cantidad internacionalmente formateada monetaria.

**char frac_digits**

El número de dígitos fraccionarios (aquéllos después de la coma decimal) para ser mostrado en una cantidad formateada monetaria.

**char p_cs_precedes**

Asignado **1** ó **0** a **currency_symbol** respecitvamente precede o sucede el valor para una cantidad formateada monetaria no negativa.

**char p_sep_by_space**

Asignado **1** ó **0** a **currency_symbol** respecitvamente está o no está separado por un espacio del valor para una cantidad formateada monetaria no negativa.

**char n_cs_precedes**

Asignado **1** ó **0** a **currency_symbol** respectivamente precede o sucede el valor para una cantidad formateada monetaria negativa.

**char n_sep_by_space**

Asignado **1** ó **0** a **currency_symbol** respecitvamente está o no está separado por un espacio del valor para una cantidad formateada monetaria negativa.

**char p_sign_posn**

Asignado un valor indicando la posición del signo positivo (**positive_sign**) para una cantidad formateada monetaria no negativa.

**char n_sign_posn**

Asignado un valor indicando la posición del signo negativo (**negative_sign**) para una cantidad formateada monetaria negativa.

Los elementos de **grouping** y **mon_grouping** son interpretados según lo siguiente:

| | |
|---|---|
| CHAR_MAX | No se realiza más agrupaciones |
| **0** | El elemento anterior ha de ser usado repetidamente para el resto de los dígitos |

| | |
|---|---|
| *otro* | El valor entero es el número de dígitos que forman parte del grupo actual. El elemento posterior es examinado para determinar el tamaño del siguiente grupo de dígitos antes del grupo actual. |

Los valores de **p_sign_posn** y **n_sign_posn** son interpretados según los siguiente:

**0** Los paréntesis rodean la cantidad y **current_symbol**

**1** La cadena de signo precede la cantidad y **current_symbol**

**2** La cadena de signo sucede la cantidad y **current_symbol**

**3** La cadena de signo inmediatamente precede la cantidad y **current_symbol**

**4** La cadena de signo inmediatamente sucede la cantidad y **current_symbol**

## Valor de retorno:

La función *localeconv* retorna un puntero al objeto rellenado. La estructura apuntado por el valor retornado no será modificado por el programa, pero puede ser sobreescrito por una llamada posterior a la función.*localeconv*. Además, llamadas a la función setlocale con categorías LC_ALL, LC_MONETARY, o LC_NUMERIC pueden sobreescribir el contenido de la estructura.

## Ejemplo:

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

int main()
{
  struct lconv *lcPtr;
  char *s;

  printf( "Localidad: \"%s\"\n\n", setlocale( LC_ALL, "C" ) );

  lcPtr = localeconv();

  printf( "decimal_point = \"%s\"\t\t",   lcPtr->decimal_point );
  printf( "thousands_sep = \"%s\"\n",     lcPtr->thousands_sep );
  printf( "grouping = \"%s\"\t\t\t",      lcPtr->grouping );
  printf( "int_curr_symbol = \"%s\"\n",   lcPtr->int_curr_symbol );
  printf( "currency_symbol = \"%s\"\t\t", lcPtr->currency_symbol );
  printf( "mon_decimal_point = \"%s\"\n", lcPtr->mon_decimal_point );
  printf( "mon_grouping = \"%s\"\t\t",    lcPtr->mon_grouping );
  printf( "positive_sign = \"%s\"\n",     lcPtr->positive_sign );
  printf( "negative_sign = \"%s\"\n",     lcPtr->negative_sign );
  printf( "int_frac_digits = (char) %d\t",   lcPtr->int_frac_digits );
  printf( "frac_digits = (char) %d\n",       lcPtr->frac_digits );
  printf( "p_cs_precedes = (char) %d\t",     lcPtr->p_cs_precedes );
  printf( "p_sep_by_space = (char) %d\n",    lcPtr->p_sep_by_space );
  printf( "n_cs_precedes = (char) %d\t",     lcPtr->n_cs_precedes );
  printf( "n_sep_by_space = (char) %d\n",    lcPtr->n_sep_by_space );
  printf( "p_sign_posn = (char) %d\t",       lcPtr->p_sign_posn );
  printf( "n_sign_posn = (char) %d\n",       lcPtr->n_sign_posn );

  printf( "\nCambiamos algunas variables para concordar con la localidad de " );
```

```
  printf( "España:\n" );
  lcPtr->decimal_point = ",";
  lcPtr->thousands_sep = ".";
  lcPtr->grouping = "3";
  printf( "decimal_point = \"%s\"\n",     lcPtr->decimal_point );
  printf( "thousands_sep = \"%s\"\n",     lcPtr->thousands_sep );
  printf( "grouping = \"%s\"\n",          lcPtr->grouping );

  return 0;
}
```

# Función localtime ANSI C

**Librería: time**

```
struct tm *localtime(const time_t *tiempoPtr);
```

La función *localtime* convierte el tiempo en formato condensado apuntado por **tiempoPtr** en el tiempo en formato separado, expresado como el tiempo local.

## Valor de retorno:

La función *localtime* retorna un puntero al objeto.

## Ejemplo:

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

int main()
{
  time_t tiempo;
  char cad[80];
  struct tm *tmPtr;

  tiempo = time(NULL);
  tmPtr = localtime(&tiempo);
  strftime( cad, 80, "%H:%M.%S, %A de %B de %Y", tmPtr );

  printf( "La hora local es: %s\n", asctime(tmPtr) );
  printf( "La hora y fecha locales son: %s\n", cad );

  return 0;
}
```

# Función log ANSI C

**Librería: math**

```
double log(double x);
```

Calcula el logaritmo natural (o neperiano). Puede producirse un error de dominio si el argumento es negativo. Puede producirse un error de recorrido si el argumento es cero.

## Valor de retorno:

La función retorna el logaritmo natural, es decir, *ln x*.

## Ejemplo:

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

int main()
{
   double x = 6.54321;

   printf( "log( %f ) = %f\n", x, log(x) );
   return 0;
}
```

# Función log10 ANSI C

**Librería: math**

```
double log10(double x);
```

Calcula el logaritmo en base 10 de **x**. Puede producirse un error de dominio si el argumento es negativo. Puede producirse un error de recorrido si el argumento es cero.

## Valor de retorno:

La función retorna el logaritmo en base 10, es decir, *log x*.

## Ejemplo:

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

int main()
{
   double x = 6.54321;

   printf( "log10( %f ) = %f\n", x, log10(x) );
   return 0;
}
```

# Función longjmp ANSI C

**Librería: setjmp**

```
void longjmp(jmp_buf entorno, int valor);
```

Restaura el entorno guardado por la invocación más reciente de la función setjmp en la misma invocación del programa, con el argumento correspondiente **entorno**. Si no ha habido tal invocación, o si una función conteniendo la invocación de la función setjmp ha terminado la ejecución en el ínterino, el comportamiento no está definido.

Todos los objetos accesibles tienen valores como en el momento *longjmp* fue llamada, excepto que los valores de los objetos de duración de almacenaje automático que son locales a la función conteniendo la invocación de la función correspondiente setjmp que no son del tipo **volatile** y han sido cambiados entre la invocación de setjmp y la invocación *longjmp* son indeterminados.

Mientras se salta las llamadas usuales a funciones y los mecanismos de retorno, *longjmp* ejecutará correctamente en contexto de interruptores, señales, y cualquiera de sus funciones asociadas. Sin embargo, si la función *longjmp* es invocada desde un controlador de señales anidado (esto es, desde una función invocada como el resultado de una señal generada durante el proceso de otra señal), el comportamiento no está definido.

Después de que *longjmp* es completada, la ejecución del programa continúa como si la invocación correspondiente de la función setjmp acabara de retornar el valor especificado por **valor**. La función *longjmp* no puede causar la función setjmp retornar el valor 0; si **valor** es 0, la función setjmp retorna el valor de 1.

## Valor de Retorno:

Esta función no retorna ningún valor.

## Ejemplo:

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

void salto( jmp_buf saltimbanqui, int v )
{
   longjmp( saltimbanqui, v );
}

int main()
{
   int valor;
   jmp_buf entorno;

   printf( "longjmp y setjmp son una forma de simular el \'goto\'\n\n" );
```

```
   valor = setjmp( entorno );   /* Volveremos aqu¡ */
   if( valor != 0 )
   {
      printf( "Longjmp con el valor: %d\n", valor );
      exit( valor );
   }

   printf( "Vamos a saltar ... \n" );
   salto( entorno, 1 );         /* Salta al momento de setjmp() */

   return 0;
}
```

# Función malloc ANSI C

**Librería: stdlib**

```
void *malloc(size_t tamanyo);
```

Adjudica espacio para un objeto, cuyo tamaño es especificado por **tamanyo** y cuyo valor es indeterminado.

## Valor de retorno:

La función *malloc* retorna un puntero nulo o un puntero al espacio adjudicado.

## Ejemplo:

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

int main( void )
{
   int *numPtr, i;
   size_t tamanyo=0;

   printf( "Introduzca el tamaño de la lista: " );
   scanf( "%d", &tamanyo );

   puts( "Adjudicamos espacio a la lista." );
   numPtr = (int *)malloc( tamanyo*sizeof(int) );

   for( i=0; i<tamanyo-1; i++ )
      printf( "%d, ", numPtr[i] = rand() % 100 + 1 );
   printf( "%d\n", numPtr[i] = rand() % 100 + 1 );

   numPtr = (int *)realloc( numPtr, tamanyo/=2 );
   printf( "Recortamos la lista a la mitad: %d\n", tamanyo );
   for( i=0; i<tamanyo-1; i++ )
      printf( "%d, ", numPtr[i] );
   printf( "%d\n", numPtr[i] );

   puts( "Liberamos el espacio." );
   free( numPtr );
```

```
    return 0;
}
```

# Función mblen ANSI C

**Librería: stdlib**

```
int mblen(const char *cad, size_t n);
```

Si **cad** no es un puntero nulo, la función *mblen* determina el número de bytes contenidos en el carácter multibyte apuntado por **cad**. Si **cad** es un puntero nulo, la función *mblen* retorna un valor distinto a cero o cero, si los códigos del carácter multibyte, respectivamente, pueden ser o no ser codificados.

## Valor de retorno:

Si **cad** no es un puntero nulo, la función *mblen* retorna o bien 0 (si **cad** apunta a un carácter nulo), o el número de bytes que son contenidos en el carácter multibyte (si los siguientes **n** o menores bytes forman un carácter multibyte válido), o **-1** (si no forman un carácter multibyte válido).

## Ejemplo:

```
/* Ejemplo sacado de la ayuda de Borland */

#include <stdlib.h>
#include <stdio.h>

int main()
{
  int     x;
  char     *mbchar    = (char *)calloc(1, sizeof( char));
  wchar_t  wchar      = L'a';
  wchar_t  *pwcnull   = NULL;
  wchar_t  *pwchar    = (wchar_t *)calloc(1,  sizeof( wchar_t ));

  printf( "Convertir un carácter ancho a un carácter multibyte:\n" );
  x = wctomb( mbchar, wchar);
  printf( "\tCaracteres convertidos: %u\n", x );
  printf( "\tCarácter multibyte: %x\n\n", mbchar );

  printf( "Tamaño del carácter multibyte (según mblen): %u\n", mblen(mbchar,
MB_CUR_MAX) );
  printf( "Convertir carácter multibyte de nuevo a un carácter ancho:\n" );

  x = mbtowc( pwchar, mbchar, MB_CUR_MAX );
  printf( "\tBytes convertidos: %u\n", x );
  printf( "\tCarácter ancho: %x\n\n", pwchar );

  printf( "Intentar convertir cuando el destinatario es nulo (NULL)\n" );
  printf( " retorna la longitud del carácter multibyte: " );
  x = mbtowc( pwcnull, mbchar, MB_CUR_MAX );
  printf( "%u\n\n", x );

  printf( "Intenta convertir un puntero nulo (NULL) a un" );
  printf( " carácter ancho:\n" );
  mbchar = NULL;
```

```
  x = mbtowc( pwchar, mbchar, MB_CUR_MAX );

  printf( "\tBytes convertidos: %u\n", x );

  return 0;
}
```

# Función mbstowcs ANSI C

**Librería: stdlib**

```
size_t mbstowcs(wchar_t *wcharsPtr, const char *cad, size_t n);
```

La función *mbstowcs* convierte una secuenca de caracteres multibyte que empiezan en el estado inicial de traslado del array apuntado por **cad** en una secuencia códigos correspondientes y guarda no más de **n** códigos en un array apuntado por **wcharsPtr**. Ninguno de los caracteres multibyte que siguen después de un carácter nulo (el cual es convertido en un código con valor cero) será examinado o convertido. Cada carácter multibyte es convertido como si hubiese llamado a la función mbtowc, excepto que el estado de traslado de la función mbtowc no es afectado.

No más de **n** elementos serán modificados en el array apuntado por **wcharsPtr**. El comportamiento de copiado entre objetos que se superimponen no está definido.

## Valor de retorno:

Si un carácter multibyte es encotrado, la función *mbstowcs* retorna (**size_t)-1**. De lo contrario, la función *mbstowcs* retorna el número de elementos modificados del array, sin incluir un código para el carácter nulo, si existe.

## Ejemplo:

```
/* Ejemplo sacado de la ayuda de Borland */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int x;
  char    *mbst = (char *)malloc(MB_CUR_MAX);
  wchar_t *pwst = L"Hi";
  wchar_t *pwc  = (wchar_t *)malloc(sizeof( wchar_t));

  printf( "MB_CUR_MAX = %d\n\n", MB_CUR_MAX );

  printf( "Convertir a una cadena multibyte:\n" );
  x = wcstombs( mbst, pwst, MB_CUR_MAX );
  printf( "\tCaracteres convertidos %u\n", x );
  printf( "\tValor hexadecimal del primer" );
  printf( " carácter multibyte: %#.4x\n\n", mbst );

  printf( "Convertir de nuevo a una cadena de caracteres anchas:\n" );

  x = mbstowcs( pwc, mbst, MB_CUR_MAX );
```

```
  printf( "\tCaracteres convertidos: %u\n",x );
  printf( "\tValor hexadecimal del primer" );
  printf( " carácter ancho: %#.4x\n\n", pwc );

  return 0;
}
```

# Función mbtowc ANSI C

**Librería: stdlib**

```
int mbtowc(wchar_t *wcharPtr, const char *cad, size_t n);
```

Si **cad** no es un puntero nulo, la función *mbtowc* determina el número de bytes contenidos en el carácter multibyte apuntado por **cad**. Entonces determina el código para el valor del tipo wchar_t que corresponde a un carácter multibyte. (El valor del código correspondiendo al carácter nulo es cero). Si el car0cter multibyte es válido y **wcharPtr** no es un puntero nulo, la función *mbtowc* guarda el código en el objeto apuntado por **wcharPtr**. Al menos **n** bytes del array apuntado por **cad** serán examinados.

## Valor de retorno:

Si **cad** es un puntero nulo, la función *mblen* retorna un valor distinto a cero o cero, si los códigos del carácter multibyte, respectivamente, pueden ser o no ser codificados. Si **cad** no es un puntero nulo, la función *mblen* retorna o bien 0 (si **cad** apunta a un carácter nulo), o el número de bytes que son contenidos en el carácter multibyte (si los siguientes **n** o menores bytes forman un carácter multibyte válido), o **-1** (si no forman un carácter multibyte válido). En ningún caso, el valor retornado será mayor que **n** o el valor de la macro MB_CUR_MAX.

## Ejemplo:

```
/* Ejemplo sacado de la ayuda de Borland */

#include <stdlib.h>
#include <stdio.h>

int main()
{
  int     x;
  char     *mbchar    = (char *)calloc(1, sizeof( char));
  wchar_t  wchar      = L'a';
  wchar_t  *pwcnull   = NULL;
  wchar_t  *pwchar    = (wchar_t *)calloc(1,  sizeof( wchar_t ));

  printf( "Convertir un carácter ancho a un carácter multibyte:\n" );
  x = wctomb( mbchar, wchar);
  printf( "\tCaracteres convertidos: %u\n", x );
  printf( "\tCarácter multibyte: %x\n\n", mbchar );

  printf( "Tamaño del carácter multibyte (según mblen): %u\n", mblen(mbchar,
MB_CUR_MAX) );
  printf( "Convertir carácter multibyte de nuevo a un carácter ancho:\n" );

  x = mbtowc( pwchar, mbchar, MB_CUR_MAX );
  printf( "\tBytes convertidos: %u\n", x );
  printf( "\tCarácter ancho: %x\n\n", pwchar );

  printf( "Intentar convertir cuando el destinatario es nulo (NULL)\n" );
  printf( " retorna la longitud del carácter multibyte: " );
  x = mbtowc( pwcnull, mbchar, MB_CUR_MAX );
```

```
  printf( "%u\n\n", x );

  printf( "Intenta convertir un puntero nulo (NULL) a un" );
  printf( " carácter ancho:\n" );
  mbchar = NULL;
  x = mbtowc( pwchar, mbchar, MB_CUR_MAX );

  printf( "\tBytes convertidos: %u\n", x );

  return 0;
}
```

# Función memchr ANSI C

**Librería: string**

```
void *memchr(const void *s, int c, size_t n);
```

Localiza la primera aparición del carácter **c** (convertido a **unsigned char**) en los primeros **n** caracteres (cada uno interpretado como un **unsigned char**) del objeto apuntado por **s**.

## Valor de retorno:

La función retorna un puntero al carácter localizado, o un puntero nulo si el carácter no apareció en el objeto.

## Ejemplo:

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

int main()
{
   char cadena[] = "Erase una vez...";
   char *puntero;

   puntero = (char *)memchr( cadena, 'a', 5 );
   printf( "%s\n", cadena);
   printf( "%s\n", puntero );

   return 0;
}
```

# Función memcmp ANSI C

**Librería: string**

```
int memcmp(const void *s1, const void *s2, size_t n);
```

Compara los primeros **n** caracteres del objeto apuntado por **s1** (interpretado como **unsigned char**) con los primeros **n** caracteres del objeto apuntado por **s2** (interpretado como **unsigned char**).

## Valor de retorno:

La función retorna un número entero mayor, igual, o menor que cero, apropiadamente según el objeto apuntado por **s1** es mayor, igual, o menor que el objeto apuntado por **s2**.

## Ejemplo:

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

int main()
{
   char a[3] = { 82, 81, 84 };
   char b[3] = { 85, 83, 86 };
   int i;

   for( i=0; i<3; i++ )
     printf( "a[%d]=%c ", i, a[i] );
   printf( "\n" );
   for( i=0; i<3; i++ )
     printf( "b[%d]=%c ", i, b[i] );
   printf( "\n" );

   i = memcmp( a, b, 4 );
   printf( "a es " );
   if( i < 0 )  printf( "menor que" );
   else if( i > 0 )  printf( "mayor que" );
   else  printf( "igual a" );
   printf( " b\n" );

   return 0;
```

```
}
```

# Función memcpy ANSI C

**Librería: string**

```
void *memcpy(void *s1, const void *s2, size_t n);
```

Copia los primeros **n** caracteres del objeto apuntado por **s2** al objeto apuntado por **s1**.

## Valor de retorno:

La función retorna el valor de **s1**. Si al copiar un objeto al otro se superponen, entonces el comportamiento no está definido.

## Ejemplo:

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

int main()
{
   char a[7] = "abcdefg";
   char *ptr;
   int i;

   memcpy( ptr, a, 5 );
   for( i=0; i<7; i++ )
     printf( "a[%d]=%c ", i, a[i] );
   printf( "\n" );
   for( i=0; i<5; i++ )
     printf( "ptr[%d]=%c ", i, ptr[i] );
   printf( "\n" );

   return 0;
}
```

# Función memmove ANSI C

**Librería: string**

```
void *memmove(void *s1, const void *s2, size_t n);
```

Copia los primeros **n** caracteres del objeto apuntado por **s2** al objeto apuntado por **s1**.Sin embargo, se asegura de que no estén superpuestos. Por esta razón, copia los caracteres a un array/arreglo temporalmente. Después vuelve a copiar del array temporal al objeto en cuestión.

## Valor de retorno:

La función retorna el valor de **s1**.

## Ejemplo:

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

int main()
{
   char a[7] = "abcdefg";
   char *ptr;
   int i;

   memmove( ptr, a, 5 );
   for( i=0; i<7; i++ )
     printf( "a[%d]=%c ", i, a[i] );
   printf( "\n" );
   for( i=0; i<5; i++ )
     printf( "ptr[%d]=%c ", i, ptr[i] );
   printf( "\n" );

   return 0;
}
```

# Función memset ANSI C

**Librería: string**

```
void *memset(void *s, int c, size_t n);
```

Copia el valor de **c** (convertido a **unsigned char**) en cada uno de los primeros **n** caracteres en el objeto apuntado por **s**.

## Valor de retorno:

La función retorna el valor de **s**.

## Ejemplo:

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

int main()
{
   char c = 'F';
   char *s;
   int i;

   s = (char*)malloc(5*sizeof(char));
   memset( s, c, 5 );
   for( i=0; i<5; i++ )
     printf( "c[%d]=%c ", i, c );
   printf( "\n" );
   free(s);
   return 0;
}
```

# Función mktime ANSI C

**Librería: time**

```
time_t mktime(struct tm *tiempoPtr);
```

La función *mktime* convierte el tiempo en formato separado, expresado como el tiempo local, en la estructura apuntada por **tiempoPtr** en un valor de tiempo en formato condensado con el mismo código que el valor retornado por la función time. Los valores originales de los componentes **tm_wday** y **tm_yday** de la estructura son ignorados., y los valores originales de los otros componentes no están restringidos a los intervalos indicados en la sección de struct tm. Cuando la función acabe con éxito, los valores de los componentes **tm_wday** y **tm_yday** de la estructura son asignados apropiadamente, y los otros componentes son asignados para representar el tiempo especificado, pero con sus valores forzados para los intervalos indicados en la sección de struct tm; el último valor de **tm_day** no es asignado hasta que **tm_mon** y **tm_year** son determinados.

## Valor de retorno:

La función *mktime* retorna el tiempo especificado codificado como un valor de tipo time_t. Si el tiempo no puede ser representado, la función retorna el valor **(time_t)-1**.

## Ejemplo:

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

int main( void )
{
   int segundos=0;
   time_t *actualPtr, alarma;
   struct tm *alarmaPtr;

   printf( "Introduzca los segundos en el futuro para la alarma: " );
   scanf( "%d", &segundos );

   *actualPtr = time( NULL );
   alarmaPtr = localtime( actualPtr );
   alarmaPtr->tm_sec += segundos;
   alarma = mktime( alarmaPtr );

   printf( "La hora local: %s\n", ctime(actualPtr) );

   while( difftime( alarma, *actualPtr ) > 0 )
      *actualPtr = time( NULL );

   printf( "ALARMA!!!\n\n" );
```

```
    printf( "La hora local: %s\n", ctime(actualPtr) );

    return 0;
}
```

# Función modf ANSI C

**Librería: math**

```
double modf(double valor, double *iptr);
```

Parte en dos el argumento **valor** en una parte entera y otra decimal, cada una de las cuales tiene el mismo signo que el argumento. Guarda la parte entera como un **double** en el objeto apuntado por **iptr**.

## Valor de retorno:

La función modf retorna la parte decimal con signo de **valor**.

## Ejemplo:

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

int main()
{
   double valor = 6.54321, *iptr, resultado;

   resultado = modf( valor, iptr );
   printf( "modf( %f, %f ) = %f\n", valor, *iptr, resultado );
   return 0;
}
```

# Función perror ANSI C

**Librería: stdio**

```
int perror(char *cadena);
```

La función *perror* transforma el número de error en la expresión entera de errno a un mensaje de error. Escribe una secuencia de caracteres al stream estándar de errores, esto es: primero (si **cadena** no es un puntero nulo y el carácter apuntado por **cadena** no es el carácter nulo), la cadena apuntada por **cadena** seguido de dos puntos (:) y un espacio; entonces un mensaje de errores apropiado seguido por un carácter de línea nueva. El contenido de los mensajes de errores es el mismo que aquello retornado por la función strerror con el argumento errno, que están definidos según la implementación.

## Valor de retorno:

La función *perror* no retorna ningún valor.

## Ejemplo:

```
#include <stdio.h>

int main( void )
{
   FILE *fichero;
   char nombre[13]="noexiste.dat", error[81];

   fichero = fopen( nombre, "r" );
   perror( error );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "existe (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      puts( error );
      return 1;
   }

   while( !feof(fichero) )
   {
      fgetc( fichero );
      tamanyo++;
   }
   printf( "El fichero \'%s\' contiene %d caracteres.\n", nombre, tamanyo );

   if( !fclose(fichero) )
      printf( "Fichero cerrado\n" );
   else
   {
      perror( error );
      printf( "Error: fichero NO CERRADO\n" );
```

```
        puts( error );
        return 1;
    }

    return 0;
}
```

# Función pow ANSI C

**Librería: math**

```
double pow(double x, double y);
```

Calcula **x** elevado a la potencia de **y**. Puede producirse un error de dominio si **x** es negativo e **y** no es un valor entero. También se produce un error de dominio si el resultado no se puede representar cuando **x** es cero e **y** es menor o igual que cero. Un error de recorrido puede producirse.

## Valor de retorno:

La función pow retorna el resultado de $x^y$.

## Ejemplo:

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

int main()
{
   double x = 6.54321, y = 0.56789;

   printf( "pow( %f, %f ) = %f\n", x, y, pow(x,y) );
   return 0;
}
```

# Función printf ANSI C

**Librería: stdio**

```
int printf(char *cadena, const char *formato, ...);
```

Esta función es equivalente a fprintf, con el argumento stdout interpuesto antes de los argumentos a *printf*.

## Valor de retorno:

La función *printf* retorna el número de caracteres transmitidos, o un valor negativo si se produce un error de salida.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[20];
   unsigned int edad=0;

   printf( "Escriba su nombre: " );
   scanf( "%s", nombre );
   printf( "Escriba su edad: " );
   scanf( "%u", &edad );

   fprintf( stdout, "\nHola %s. Tienes %d anyos.\n", nombre, edad );

   return 0;
}
```

# Macro putc ANSI C

**Librería: stdio**

```
int putc(int c, FILE *stream);
```

Esta función es equivalente a fputc, excepto que si es implementado como una macro, puede evaluar **stream** más de una vez - el argumento debería ser una expresión sin efectos secundarios.

## Valor de retorno:

La función *putc* retorna el carácter escrito. Si ocurre un error de escritura, el indicador de error para el stream es activado y *putc* retorna EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[11]="datos2.dat";
   FILE *fichero;

   fichero = fopen( nombre, "w" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   for( i=0; i<25; i++ )   putc( i+'a', fichero );

   if( !fclose(fichero) )
      printf( "\nFichero cerrado\n" );
   else
   {
      printf( "\nError: fichero NO CERRADO\n" );
      return 1;
```

```
    }

    return 0;
}
```

# Función putchar ANSI C

**Librería: stdio**

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

Esta función es equivalente a putc con el segundo argumento stdout.

## Valor de retorno:

La función *putchar* retorna el carácter escrito. Si ocurre un error de escritura, el indicador de error para el stream es activado y *putchar* retorna EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[20]="";
   int i;

   printf( "Escribe tu nombre: " );
   scanf( "%s", nombre );

   printf( "Tu nombre, al reves:\n" );
   for( i=strlen(nombre)-1; i>=0; i-- )
      putchar( nombre[i] );
   printf( "\n" );

   return 0;
}
```

# Función puts ANSI C

**Librería: stdio**

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

Esta función escribe la cadena apuntado por **cadena** en el stream apuntado por stdout, y añade un carácter de línea nueva a la salida. El carácter nulo final no es escrito.

## Valor de retorno:

La función *puts* retorna EOF si ocurre un error de escritura; si no, entonces retorna un valor no negativo.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char oracion[47] = "Esto es un ejemplo para usar la funcion \'puts\'";

   if( puts( oracion ) > 0 )   printf( "\nEl ejemplo ha funcionado correctamente\n"
);
   else   printf( "\nERROR: La funcion no se ha realizado correctamente\n" );

   return 0;
}
```

# Función qsort ANSI C

**Librería: stdlib**

```
void qsort(void *base, size_t nmemb, size_t tamanyo,
   int (*comparar)(const void *, const void *));
```

Ordena un array de **nmemb** objetos. El elemento inicial es apuntado por **base**. El tamaño de cada elemento del array está especificado por **tamanyo**. El contenido del array es ordenado en el orden de ascienso según una función de comparación apuntada por **comparar**, la cual es llamada con dos argumentos que apuntan a los objetos a ser comparados. La función retornará un entero menor, igual, o mayor que cero si el primer objeto es considerado, respectivamente a ser menor, igual, o mayor que el segundo. Si los dos elementos son iguales, su orden en el array ordenado no está definido.

## Valor de retorno:

La función *qsort* no retorna ningún valor.

## Ejemplo:

```
/* Implementación del algoritmo quick-sort para la ordenación de vectores
** y bsearch para buscar ciertos elementos
*/

#include <stdlib.h>
#include <stdio.h>

#define ELEMENTOS 100

/* Prototipo de la función de comparar */
int comparar(const void *arg1, const void *arg2);
{
 if(*(int *)arg1 < *(int *)arg2) return -1;
   else if(*(int *)arg1 > *(int *)arg2) return 1;
   else return 0;
}

int main()
{
   int i, num;
   int lista[ELEMENTOS], int *elementoPtr;

   /* Contenido aleatorio */
   for(i = 0; i < ELEMENTOS; i++) lista[i] = rand() % 100 + 1;

   /* Quick-Sort */
   qsort(lista, ELEMENTOS, sizeof(lista[0]), comparar);

    /* Mostramos la lista ordenada */
   for(i = 0; i < ELEMENTOS; i++) printf("%d ", lista[i]);
   printf("\n");

   /* Ahora busquemos algunos elementos en la lista */
   puts( "Especifique un numero a encontrar dentro de la lista ordenada\n(un numero
negativo para salir):" );
   scanf( "%d", &num );
   while( num >= 0 )
```

```
   {
      if( (elementoPtr = bsearch( num, lista, ELEMENTOS, sizeof(lista[0]), comparar
)) != NULL )
          printf( "Encontrado: %d\n", *elementoPtr );
      else
          printf( "No encontrado: %d\n", num );
      scanf( "%d", &num );
   }

   return 0;
}
```

# Función raise ANSI C

**Librería: signal**

```
int raise(int señal);
```

La función envía la señal **señal** al programa en ejecución.

## Valor de retorno:

La función *raise* retorna cero, si la operación fue realizada con éxito, y un valor distinto a cero, si falla.

## Ejemplo:

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

int main()
{
   printf( "Iniciando el programa...\n" );
   printf( "Cerrando el programa...\n" );
   if( !raise( SIGTERM ) )
      printf( "Cierre conseguido...\n" );
   else
      printf( "Peticion de cierre: no satisfecha...\n" );

   printf( "\nAdios\n\n" );

   return 0;
}
```

# Función rand ANSI C

**Librería: stdlib**

```
int rand(void);
```

La función *rand* calcula una secuencia de números enteros pseudo-aleatorios en el intervalo de 0 á RAND_MAX.

## Valor de retorno:

La función *rand* retorna un entero pseudo-aleatorio.

## Ejemplo:

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

int main()
{
   unsigned int i=1;

   printf( "30 numeros generados aleatoriamente: \n\n" );
   for( i=1; i<30; i++ )
      printf( "%d, ", rand() );
   printf( "%d\n", rand() );

   return 0;
}
```

# Función realloc ANSI C

**Librería: stdlib**

```
void *realloc(void *ptr, size_t tamanyo);
```

Cambia el tamaño del objeto apuntado por **ptr** al tamaño especificado por **tamanyo**. El contenido del objeto no cambiará hasta el menor de los tamaños nuevo y viejo. Si el tamaño nuevo es mayor, el valor de la porción nuevamente adjudicada del objesto es indeterminado. Si **ptr** es un puntero nulo, la función *realloc* se comporta a igual que la función malloc para el tamaño especificado. De lo contrario, si **ptr** no es igual a un puntero previamente retornado por la función calloc, malloc, o *realloc*, o si el espacio ha sido desadjudicado por una llamada a la función free, o *realloc*, el comportamiento no está definido. Si el espacio no puede ser desadjudicado, el objeto apuntado por **ptr** no varía. Si **tamanyo** es cero y **ptr** no es nulo, el objeto al que apunta es liberado.

## Valor de retorno:

La función *realloc* retorna o bien un puntero nulo o bien un puntero posiblemente al espacio adjudicado mudado.

## Ejemplo:

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

int main( void )
{
   int *numPtr, i;
   size_t tamanyo=0;

   printf( "Introduzca el tamaño de la lista: " );
   scanf( "%d", &tamanyo );

   puts( "Adjudicamos espacio a la lista." );
   numPtr = (int *)malloc( tamanyo*sizeof(int) );

   for( i=0; i<tamanyo-1; i++ )
      printf( "%d, ", numPtr[i] = rand() % 100 + 1 );
   printf( "%d\n", numPtr[i] = rand() % 100 + 1 );

   numPtr = (int *)realloc( numPtr, tamanyo/=2 );
```

```
   printf( "Recortamos la lista a la mitad: %d\n", tamanyo );
   for( i=0; i<tamanyo-1; i++ )
      printf( "%d, ", numPtr[i] );
   printf( "%d\n", numPtr[i] );

   puts( "Liberamos el espacio." );
   free( numPtr );

   return 0;
}
```

# Función remove ANSI C

**Librería: stdio**

```
int remove(const char *nombre);
```

El nombre del fichero apuntado por la cadena **nombre** ya no es accesible por ese nombre. Cualquier intento posterior a abrir el fichero usando ese nombre fallará, al menos que se cree de nuevo. Si el fichero está abierto el comportamiento de la función está definido según la implementación del compilador. Por esta razón se ha de cerrar antes de borrar.

## Valor de retorno:

La función retorna cero si la operación fue realizada con éxito. Si falla, entonces retorna un valor cualquiera distinto a cero.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[24] = "fichero_para_borrar.tmp";
   FILE *fichero;

   fichero = fopen( nombre, "r" );    /* El fichero ha de existir primeramente */
   printf( "fichero: %s", nombre );
   if( fichero != NULL )
   {
      fclose( fichero );
      if( remove(nombre) == 0 )   printf( "\nBorrado\n" );
      else   printf( "\nNo pudo ser borrado\n" );
   }
   else   printf( ", no encontrado\n" );

   return 0;
}
```

# Función rename ANSI C

**Librería: stdio**

```
int rename(const char *viejo, const char *nuevo);
```

El nombre del fichero apuntado por la cadena **viejo** será conocido como el nombre apuntado por la cadena **nuevo**. Cualquier intento posterior a abrir el fichero usando ese nombre fallará, al menos que se cree de nuevo. Si el fichero nombrado por **nuevo** ya existe anteriormente a la llamada de *rename*, el comportamiento de la función está definido según la implementación del compilador.

## Valor de retorno:

La función retorna cero si la operación fue realizada con éxito. Si falla, entonces retorna un valor cualquiera excepto cero; en este caso el fichero sigue siendo conocido con el nombre **viejo**.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char viejo[18] = "fichero_viejo.tmp", nuevo[18] = "fichero_nuevo.tmp";

   printf( "fichero viejo: %s", viejo );
   if( rename(viejo, nuevo) == 0 )   printf( ", renombrado: %s\n", nuevo );
   else   printf( "\nNo pudo ser renombrado\n" );

   return 0;
}
```

# Función rewind ANSI C

**Librería: stdio**

```
void rewind(FILE *stream);
```

La función *rewind* coloca el indicador de posición de fichero para el stream apuntado por **stream** al comienzo del fichero. Es equivalente a **(void) fseek( stream, 0L, SEEK_SET )** excepto que el indicador de errores para el stream es despejado.

## Valor de retorno:

La función *rewind* no retorna ningún valor.

## Ejemplo:

```
#include <stdio.h>

int main()
{
  char nombre[11] = "datos4.dat",
       mensaje[81]="Esto es nua rpueba usando fgetpos y fsetpos.";
  FILE *fichero;
  fpos_t posicion=0, comienzo;

  fichero = fopen( nombre, "w+" );
  printf( "Fichero: %s -> ", nombre );
  if( fichero )
    printf( "creado (ABIERTO)\n" );
  else
  {
    printf( "Error (NO ABIERTO)\n" );
    return 1;
  }

  fgetpos( fichero, &comienzo );
  printf( "Posicion del fichero: %d\n", posicion );

  fprintf( fichero, mensaje );
  printf( "\nEscrito: \"%s\"\n", mensaje );

  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  fsetpos( fichero, &comienzo );
  fprintf( fichero, "%s", "Esto es una prueba" );
  printf( "Corregiendo errores...Escrito: \"Esto es una prueba\"\n" );
```

```
  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  rewind( fichero );
  printf( "\"Rebobinando\" el fichero -> Vuelta al comienzo\n" );
  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  printf( "\nLeyendo del fichero \"%s\"\n", nombre );
  fgets( mensaje, 81, fichero );
  printf( "\"%s\"\n\n", mensaje );

  fgetpos( fichero, &posicion );
  printf( "Posicion del fichero: %d\n", posicion );

  if( !fclose(fichero) )
    printf( "Fichero cerrado\n" );
  else
  {
    printf( "Error: fichero NO CERRADO\n" );
    return 1;
  }

  return 0;
}
```

# Función scanf ANSI C

**Librería: stdio**

```
int scanf(const char *formato, ...);
```

Esta función es equivalente a fscanf con el argumento stdin interpuesto antes de los argumentos a *scanf*.

## Valor de retorno:

La función *scanf* retorna el número de datos de entrada asignados, que puede ser menor que ofrecido, incluso cero, en el caso de un error de asignación. Si un error de entrada ocurre antes de cualquier conversión, la función fscanf retorna el valor de la macro EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[20]="";
   unsigned int edad=0;

   printf( "Escriba su nombre: " );
   scanf( "%s", nombre );
   printf( "Escriba su edad: " );
   fscanf( stdin, "%u", &edad );

   printf( "\nHola %s. Tienes %d anyos.\n", nombre, edad );

   return 0;
}
```

# Función setbuf ANSI C

**Librería: stdio**

```
void setbuf(FILE *stream, char *acumulador);
```

Esta función es equivalente a la función setvbuf pasando los valores _IOFBF para **modo** y BUFSIZ para **tamaño**, o (si **acumulador** es un puntero nulo), con el valor _IONBF para **modo**.

stdout y stdin no son almacenados si no son redirigidos; de lo contrario, son completamente almacenados. "No Almacenados" quiere decir que los caracteres escritos a un stream son inmediatamente transmitidos al fichero o dispostivo, mientras que "Almacenados" significa que los caracteres son acumulados y escritos como un bloque de información. La función *setbuf* produce resultados impredecibles al menos que que sea llamado después de abrir **stream** o después de llamar a fseek. Llamando a *setbuf* después de **stream** no ha sido almacenado es legal y no causará problemas.

## Valor de retorno:

La función *setbuf* no retorna ningún valor.

## Ejemplo:

```
#include <stdio.h>

int main()
{
  char acumulador[BUFSIZ];

  setbuf( stdout, acumulador );

  printf( "Esto es una prueba\n" );
  printf( "Este mensaje se mostrara a la vez\n" );
  printf( "setbuf, acumula los datos en un puntero\n" );
  printf( "hasta que se llene completamente\n" );

  fflush( stdout );

  return 0;
}
```

# Función setjmp ANSI C

**Librería: setjmp**

```
int setjmp(jmp_buf entorno);
```

Guarda su entorno de llamadas en el argumento **entorno** para uso posterior por la función longjmp.

## Valor de Retorno:

Si el retorno es de una invocación directa, la función *setjmp* retorna el valor, cero. Si el retorno es desde una llamada de la función longjmp, la función *setjmp* retorna un valor distinto a cero.

Una invocación de la función *setjmp* aparecerá slamente en uno de los siguientes contextos:

- La expresión controladora completa de una sentencia de selección o iteración.
- Un operando de un operador de relación o igualdad con el otro operando una expresión constante y entera, con la expresión resultante siendo la expresión controladora completa de una sentencia de selección o iteración.
- El operando de un operador unario **!** con la expresión resultante siendo la expresión controladora completa de una sentencia de selección o iteración.
- La expresión completa de una sentencia de una expresión.

## Ejemplo:

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

void salto( jmp_buf saltimbanqui, int v )
{
   longjmp( saltimbanqui, v );
}

int main()
{
   int valor;
   jmp_buf entorno;

   printf( "longjmp y setjmp son una forma de simular el \'goto\'\n\n" );

   valor = setjmp( entorno );   /* Volveremos aqu¡ */
   if( valor != 0 )
   {
      printf( "Longjmp con el valor: %d\n", valor );
      exit( valor );
   }

   printf( "Vamos a saltar ... \n" );
```

```
    salto( entorno, 1 );          /* Salta al momento de setjmp() */

    return 0;
}
```

# Función setlocale ANSI C

**Librería: locale**

```
char *setlocale(int categoria, const char *localidad);
```

La función *setlocale* selecciona la porción apropiada de la localidad del programa especificado por los argumentos **categoria** y **localidad**. La función *setlocale* puede ser usada para cambiar o preguntar la localidad actual total del programa o porciones de ello. El valor de LC_ALL para **categoria** nombra la localidad total del programa; los otros valores para **categoria** nombran solamente una porción de la localidad del programa. LC_COLLATE afecta el comportamiento de las funciones strcoll y strxfrm. LC_CTYPE afecta el comportamiento de las funciones que manipulan caracteres y de las funciones de multibyte. LC_MONETARY afecta a la información de formato monetario retornada por la función localeconv. LC_NUMERIC afecta el carácter de la coma decimal para las funciones de formato de entrada/salida, las funciones de conversión de cadenas, y de la información de formato no monetario retornada por localeconv. LC_TIME afecta el comportamiento de strftime.

Un valor de **"C"** para **localidad** especifica el entorno mínimo para la traducción de C; un valor de **""** para **localidad** especifica el entorno nativo definido según la implementación. Otras cadenas definidas según la implementación pueden ser pasadas a *setlocale*. Al comienzo de la ejecución del programa, la equivalente a **setlocale( LC_ALL, "C" );** es ejecutada.

## Valor de retorno:

Si un puntero a una cadena es dado para **localidad** y la selección puede ser llevado a cabo, la función *setlocale* retorna un puntero a la cadena asociada con la **categoria** especificada para la localidad nueva. Si la selección no se pudo realizar, la función *setlocale* retorna un puntero nulo y la localidad del programa no varía.

Un puntero nulo para **localidad** causa la función *setlocale* retornar un puntero a la cadena asociado con la **categoria** de la localidad actual del programa; la localidad actual del programa no varía.

El puntero a la cadena retornado por la función *setlocale* es tal que una llamada posterior con ese valor de la cadena y su categoría asociada restaurará la parte de la localidad del programa. La cadena puntada será modificada por el programa, pero puede ser sobreescrito por una llamada a la función *setlocale*.

## Ejemplo:

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

int main()
{
   struct lconv *lcPtr;
   char *s;

   printf( "Localidad: \"%s\"\n\n", setlocale( LC_ALL, "C" ) );

   lcPtr = localeconv();

   printf( "decimal_point = \"%s\"\t\t",   lcPtr->decimal_point );
   printf( "thousands_sep = \"%s\"\n",     lcPtr->thousands_sep );
   printf( "grouping = \"%s\"\t\t\t",      lcPtr->grouping );
   printf( "int_curr_symbol = \"%s\"\n",   lcPtr->int_curr_symbol );
   printf( "currency_symbol = \"%s\"\t\t", lcPtr->currency_symbol );
```

```
   printf( "mon_decimal_point = \"%s\"\n", lcPtr->mon_decimal_point );
   printf( "mon_grouping = \"%s\"\t\t",    lcPtr->mon_grouping );
   printf( "positive_sign = \"%s\"\n",     lcPtr->positive_sign );
   printf( "negative_sign = \"%s\"\n",     lcPtr->negative_sign );
   printf( "int_frac_digits = (char) %d\t",   lcPtr->int_frac_digits );
   printf( "frac_digits = (char) %d\n",       lcPtr->frac_digits );
   printf( "p_cs_precedes = (char) %d\t",     lcPtr->p_cs_precedes );
   printf( "p_sep_by_space = (char) %d\n",    lcPtr->p_sep_by_space );
   printf( "n_cs_precedes = (char) %d\t",     lcPtr->n_cs_precedes );
   printf( "n_sep_by_space = (char) %d\n",    lcPtr->n_sep_by_space );
   printf( "p_sign_posn = (char) %d\t",       lcPtr->p_sign_posn );
   printf( "n_sign_posn = (char) %d\n",       lcPtr->n_sign_posn );

   printf( "\nCambiamos algunas variables para concordar con la localidad de " );
   printf( "España:\n" );
   lcPtr->decimal_point = ",";
   lcPtr->thousands_sep = ".";
   lcPtr->grouping = "3";
   printf( "decimal_point = \"%s\"\n",     lcPtr->decimal_point );
   printf( "thousands_sep = \"%s\"\n",     lcPtr->thousands_sep );
   printf( "grouping = \"%s\"\n",          lcPtr->grouping );

   return 0;
}
```

# Función setvbuf ANSI C

**Librería: stdio**

```
int setvbuf(FILE *stream, char *acumulador, int modo, size_t tamanyo);
```

Esta función sólo puede ser usada después de que el stream apuntado por **stream** ha sido asociado con un fichero abierto y antes de otra operación cualquiera es llevada acabo al stream. El argumento **modo** determina cómo **stream** será almacenado según lo siguiente: _IOFBF ocasiona la entrada/salida a ser completamente almacenado; _IOLBF ocasiona la entrada/salida a almacenar por líneas; _IONBF ocasiona la entrada/salida a no ser almacenado. Si **acumulador** no es un puntero nulo, el array al que es apuntado puede ser usado en vez de la acumulación adjudicada por la función *setvbuf*. El argumento **tamanyo** especifica el tamaño del array. El contenido del array en cualquier ocasión es indeterminado.

stdout y stdin no son almacenados si no son redirigidos; de lo contrario, son completamente almacenados. "No Almacenados" quiere decir que los caracteres escritos a un stream son inmediatamente transmitidos al fichero o dispostivo, mientras que "Almacenados" significa que los caracteres son acumulados y escritos como un bloque de información.

Los tres valores de **modo** son éstos:

| | |
|---|---|
| **_IOFBF** | Cuando un acumulador esté vacío, la siguiente operación intentará llenar el acumulador completamente. En salida, el acumulador será completamente llenado antes de cualquier dato sea escrito en el fichero. |
| **_IOLBF** | Cuando un acumulador esté vacío, la siguiente operación intentará llenar el acumulador completamente. En salida, sin embargo, el acumulador será transmitido y despejado en cuanto un carácter de línea nueva sea escrito en el fichero. |
| **_IONBF** | Los parámetros **acumulador** y **tamanyo** son ignorados. Cada operación de entrada leerá directamente desde el fichero, y cada operación de salida inmediatamente escribirá los datos al fichero. |

## Valor de retorno:

La función *setvbuf* retorna cero cuando tiene éxito, o un valor cualquiera excepto cero, si un valor inválido es pasado a **modo** o si la operación no se puede llevar a cabo.

## Ejemplo:

```
#include <stdio.h>

int main()
{
  char mensaje[BUFSIZ];

  setvbuf( stderr, mensaje, _IOFBF, BUFSIZ );

  printf( "Este mensaje se puede mostrar sin problemas\n" );
  fprintf( stderr, "\nPero este mensaje solo se muestra\n" );
  fprintf( stderr, "cuando el acumulador para stderr se llene\n" );
  fprintf( stderr, "o cuando se llame a fflush\n\n" );

  fflush( stderr );

  return 0;
}
```

# Función signal ANSI C

**Librería: signal**

```
void (*signal(int señal, void (*func)(int))(int);
```

Elige una de las tres formas se ha de manejar el número de señal **señal**. Si el valor de **func** es SIG_DFL, la señal se controlará por defecto. Si el valor de **func** es SIG_IGN, la señal será ignorada. Si no, **func** apuntará a una función a ser llamada cuando una señal se produce. Dicha función se llama un *controlador de señales*.

Cuando se produce una señal, si **func** apunta a una función, primeramente la equivalente a **signal( señal, SIG_DFL );** es ejecutada o un bloqueo de señal, definido según la implementación, es realizado. (Si el valor de **señal** es SIGILL, el caso de que se reinicie SIG_DFL está definido según la implementación). A continuación la equivalente a **(*func)( señal );** es ejecutada. La función **func** puede terminar mediante ejecutando la sentencia **return** o mediante llamando las funciones abort, exit, o longjmp. Si **func** ejecuta una sentencia de retorno y el valor de **señal** era SIGFPE o cualquier otro valor definido según la implementación correspondiendo a una excepción computada, el comportamiento no está definido. De lo contrario, el programa continúa la ejecución desde el momento de interrupción.

Si la señal se produce de otra forma que el resultado de llamar las funciones {f:abort} o raise, el comportamiento no está definido si el controlador de señales llama a cualquier función de la librería estándar distinta a sí misma, la función *signal* (con el primer argumento de la señal numérica correspondiendo a la señal que causó la invocación al controlador), o hace referencia a cualquier objeto con una duración de almacenamiento estático distinto a asignando un valor a una variable de duración estática de tipo **volatile sig_atomic_t**. Además, si tal llamada a la función *signal* resulta en un retorno de SIG_ERR, el valor de errno es indeterminado.

Al comienzo del programa, la equivalente a **signal( señal, SIG_IGN );** puede ser ejecutada para algunas señales seleccionadas de una manera definida según la implementación; la equivalentea **signal( señal, SIG_DFL );** es ejecutada para todas las demás señales definidas por la implementación.

## Valor de retorno:

La función *signal* retorna el valor de **func** para la llamada más reciente a *signal* para la señal especificada, **señal**, si la operación fue realizada con éxito. Si falla, entonces retorna {m:SIG_ERR:SIG_ERR} y un valor positivo es guardado en errno.}

## Ejemplo:

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

typedef void (*funcPtr)();

int salir=0;    /* Falso */

void controlador( int *lista_Registros )
{
```

```
   int i;

   signal( SIGINT, (funcPtr)controlador );

   printf( "Error capturado\n\n" );
   for( i=0; i<11; i++ )
      printf( "\n[%d] = %d\t", i, lista_Registros[i] );
   salir=1;     /* Verdadero */
}

int main()
{
   float x, y=2.5, h=0.01;

   signal( SIGINT, (funcPtr)controlador );

   for( x=1.0; x<=100.0 && !salir; x+=h )
      printf( "%f / %f = %f\n", y, x, y/x );    /* Lista demasiada Larga */

   return 0;
}
```

# Función sin ANSI C

**Librería: math**

```
double sin(double x);
```

Calcula el seno de **x** (medido en radianes).

## Valor de retorno:

La función sin retorna el seno, en radianes.

## Ejemplo:

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

int main()
{
   double x = 3.1416/3.0;

   printf( "sin( %f ) = %f\n", x, sin(x) );
   return 0;
}
```

# Función sinh ANSI C

**Librería: math**

```
double sinh(double x);
```

Calcula el seno hiperbólico de **x**. Aparece un error de recorrido si la magnitud de **x** es demasiada grande.

## Valor de retorno:

La función sinh retorna el seno hiperbólico.

## Ejemplo:

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

int main()
{
   double x = 3.0;

   printf( "sinh( %f ) = %f\n", x, sinh(x) );
   return 0;
}
```

# Función sprintf ANSI C

**Librería: stdio**

```
int sprintf(char *cadena, const char *formato, ...);
```

Esta función es equivalente a fprintf, excepto que el argumento **cadena** especifica un array en el cual la salida generada es para ser escrita, en vez de un stream. Un carácter nulo es escrito al final de los caracteres escritos; no es contado como parte de la suma retornada. El comportamiento acerca de copiando entre objetos que se superponen no está definido.

## Valor de retorno:

La función s*printf* retorna el número de caracteres escritos al array, sin contar el carácter nulo al final.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[20], mensaje[81];
   unsigned int edad=0;

   printf( "Escriba su nombre: " );
   scanf( "%s", nombre );
   printf( "Escriba su edad: " );
   scanf( "%u", &edad );

   sprintf( mensaje, "\nHola %s. Tienes %d anyos.\n", nombre, edad );
   puts( mensaje );

   return 0;
}
```

# Función sqrt ANSI C

**Librería: math**

```
double sqrt(double x);
```

Calcula la raíz cuadrada del valor no negativo de **x**. Puede producirse un error de dominio si **x** es negativo.

## Valor de retorno:

La función sqrt retorna el resultado de la raíz cuadrada de **x**.

## Ejemplo:

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

int main()
{
   double x = 6.54321;

   printf( "sqrt( %f ) = %f\n", x, sqrt(x) );
   return 0;
}
```

# Función srand ANSI C

**Librería: stdlib**

```
void srand(unsigned int semilla);
```

Usa el argumento como una semilla para una secuencia nueva de números pseudo-aleatorios para ser retornados por llamadas posteriores a rand. Si *srand* es entonces llamada con el mismo valor semilla, la secuencia de números pseudo-aleatorios será repetida. Si rand es llamada antes de que se hayan hecho cualquier llamada a *srand*, la misma secuencia será generada como cuando *srand* fue llamada la primera vez con un valor semilla de 1.

Las siguientes funciones definen una implementación portable de rand y *srand*.

```
static unsigned long int siguiente = 1;

int rand( void )
{
  siguiente *= 1103515245 + 12345;

  return (unsigned int) (siguiente/65536) % (RAND_MAX-1);
}

void srand( unsigned int semilla )
{
  siguiente = semilla;
}
```

## Valor de retorno:

La función s*rand* no retorna ningún valor.

## Ejemplo:

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

/* Sintaxis del programa: srand <semilla> */

int main( int argc, char *argv[] )
{
   unsigned int i=1;

   srand( atoi(argv[1]) );    /* Cada vez que se ejecute el programa, una semilla del
usuario será usada */
   printf( "30 numeros generados aleatoriamente: \n\n" );
   for( i=1; i<30; i++ )
   {
      printf( "%d, ", rand() );
      srand( rand() );    /* Cada número generado vendrá de una secuencia distinta:
"más aleatorio" */
   }
   printf( "%d\n", rand() );

   return 0;
}
```

# Función sscanf ANSI C

**Librería: stdio**

```
int sscanf(const char *cadena, const char *formato,...);
```

Esta función es equivalente a fscanf, excepto que el argumento **cadena** especifica un array desde el cual la entrada es obtenida, en vez de un stream. Llegando al final de la cadena es equivalente a encontrar un final de fichero (EOF) en la función fscanf. El comportamiento acerca de copiando entre objetos que se superponen no está definido.

## Valor de retorno:

La función *scanf* retorna el número de datos de entrada asignados, que puede ser menor que ofrecido, incluso cero, en el caso de un error de asignación. Si un error de entrada ocurre antes de cualquier conversión, la función fscanf retorna el valor de la macro EOF.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char nombre[20]="", entrada[81]="";
   unsigned int edad=0;

   printf( "Escriba su nombre y edad, separados por un espacio:\n" );
   gets( entrada );
   sscanf( entrada, "%s %u", nombre, &edad );

   printf( "Has escrito: %s\n", entrada );
   printf( "Nombre: %s. Edad: %d\n", nombre, edad );

   return 0;
}
```

# Función strcat ANSI C

**Librería: string**

```
char *strcat(char*s1, const char *s2);
```

Añade una copia de la cadena apuntada por **s2** (incluyendo el carácter nulo) al final de la cadena apuntada por **s1**. El carácter inicial de **s2** sobrescribe el carácter nulo al final de **s1**.

## Valor de retorno:

La función retorna el valor de **s1**. Si la copia hace que los objetos se superpongan, entonces el comportamiento no está definido.

## Ejemplo:

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

int main()
{
   char s1[11] = "Hola ";
   char s2[6] = "amigos";

   printf( "s1=%s\t", s1 );
   printf( "s2=%s\n", s2 );
   strcat( s1, s2 );
   printf( "s1=%s\n", s1 );

   return 0;
}
```

# Función strchr ANSI C

**Librería: string**

```
char *strchr(char *s, int c);
```

Localiza la primera aparición de **c** (convertido a **unsigned char**) en la cadena apuntada por **s** (incluyendo el carácter nulo).

## Valor de retorno:

La función retorna un puntero a partir del carácter encontrado. Si no se ha encontrado el carácter, **c**, entonces retorna un puntero null.

## Ejemplo:

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

int main()
{
   char s[11] = "Hola amigos";
   char c = 'a';

   printf( "s=%s\t", s );
   printf( "c=%c\n", c );
   printf( "strchr=%s\n", strchr( s, c ) );

   return 0;
}
```

# Función strcmp ANSI C

**Librería: string**

```
int strcmp(const char *s1, const char *s2);
```

Compara la cadena apuntada por **s1** con la cadena apuntada por **s2**.

## Valor de retorno:

La función retorna un número entero mayor, igual, o menor que cero, apropiadamente según la cadena apuntada por **s1** es mayor, igual, o menor que la cadena apuntada por **s2**.

## Ejemplo:

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

int main()
{
   char s1[5] = "Abeja";
   char s2[5] = "abeja";
   int i;

   printf( "s1=%s\t", s1 );
   printf( "s2=%s\n", s2 );

   i = strcmp( s1, s2 );
   printf( "s1 es " );
   if( i < 0 )  printf( "menor que" );
   else if( i > 0 )  printf( "mayor que" );
   else  printf( "igual a" );
   printf( " s2\n" );

   return 0;
}
```

# Función strcoll ANSI C

**Librería: string**

```
int strcoll(const char *s1, const char *s2);
```

Compara la cadena apuntada por **s1** con la cadena apuntada por **s2**, ambas interpretadas acordes a la categoría LC_COLLATE de la localidad actual.

## Valor de retorno:

La función retorna un número entero mayor, igual, o menor que cero, apropiadamente según la cadena apuntada por **s1** es mayor, igual, o menor que la cadena apuntada por **s2**, cuando ambas son interpretadas apropiadamente según la localidad actual.

## Ejemplo:

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

int main()
{
   char s1[5] = "Abeja";
   char s2[5] = "abeja";
   int i;

   printf( "s1=%s\t", s1 );
   printf( "s2=%s\n", s2 );

   i = strcoll( s1, s2 );
   printf( "s1 es " );
   if( i < 0 )  printf( "menor que" );
   else if( i > 0 )  printf( "mayor que" );
   else  printf( "igual a" );
   printf( " s2\n" );

   return 0;
}
```

# Función strcpy ANSI C

**Librería: string**

```
char *strcpy(char *s1, const char *s2);
```

Copia la cadena apuntada por **s2** (incluyendo el carácter nulo) a la cadena apuntada por **s1**.

## Valor de retorno:

La función retorna el valor de **s1**. Si al copiar una cadena a la otra se superponen, entonces el comportamiento no está definido.

## Ejemplo:

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

int main()
{
   char s2[8] = "abcdefg";
   char s1[8];

   strcpy( s1, s2 );
   printf( "s2=%s\n", s2 );
   printf( "s1=%s\n", s1 );

   return 0;
}
```

# Función strcspn ANSI C

**Librería: string**

```
size_t strcspn(const char *s1, const char *s2);
```

Cuenta el número de caracteres de una subcadena inicial apuntada por **s1** que no contenga ninguno de los caracteres en la cadena apuntada por **s2**.

## Valor de retorno:

La función retorna el número de caracteres leídos de la subcadena hasta que halla alguno de los caracteres de **s2**. El carácter nulo no se cuenta.

## Ejemplo:

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

int main()
{
   char s1[13] = "Hola a todos";
   char s2[5] = "abcd";

   printf( "s1=%s\n", s1 );
   printf( "s2=%s\n", s2 );
   printf( "strcspn(s1,s2) = %d\n", strcspn( s1, s2 ) );

   return 0;
}
```

# Función strerror ANSI C

**Librería: string**

```
char *strerror(int errnum);
```

Convierte el número de error en **errnum** a un mensaje de error (una cadena de caracteres).

## Valor de retorno:

La función retorna la cadena de caracteres conteniendo el mensaje asociado con el número de error. Esta conversión y el contenido del mensaje dependen de la implementación. La cadena no será modificada por el programa, pero sí puede ser sobrescrito con otra llamada a la misma función.

## Ejemplo:

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

int main()
{
   int errnum;

   for( errnum=0; errnum<39; errnum++ )
     printf( "strerror(%d) = %s\n", errnum, strerror( errnum ) );

   return 0;
}
```

# Función strftime ANSI C

**Librería: time**

```
size_t strftime(char *cad, size_t maxtam, const char* formato, const struct tm
*tiempoPtr);
```

La función *strftime* coloca caracteres en el array apuntado por **cad** controlado por la cadena apuntada por **formato**. La cadena **formato** consiste de cero o más especificadores de conversión y caracteres multibyte ordinarios. Todos los caracteres ordinarios (incluyendo el carácter nulo terminal) son copiados sin cambiar en el array. Si se copian entre objetos superpuestos, el comportamiento no está definido. No más de **maxtam** caracteres son colocados en el array. Cada especificador de conversión es reemplazado por los caracteres apropiados descritos en la siguiente lista. Los caracteres apropiados son determinados por la categoría LC_TIME de la localidad actual y por los valores contenidos en la estructura apuntado por **tiempoPtr**.

- **%a** Es reemplazado por la abreviatura del nombre del día de la semana de la localidad
- **%A** Es reemplazado por el nombre completo del día de la semana de la localidad
- **%b** Es reemplazado por la abreviatura del nombre del mes de la localidad
- **%B** Es reemplazado por el nombre completo del mes de la localidad
- **%c** Es reemplazado por la fecha apropiada y la representación de la hora de la localidad
- **%d** Es reemplazado por el día del mes como un número decimal (**01-31**)
- **%H** Es reemplazado por la hora (reloj de 24 horas) como un número decimal (**00-23**)
- **%I** Es reemplazado por la hora (reloj de 12 horas) como un número decimal (**01-12**)
- **%j** Es reemplazado por el día del año como un número decimal (**001-366**)
- **%m** Es reemplazado por el mes como un número decimal (**01-12**)
- **%M** Es reemplazado por el minuto como un número decimal (**00-59**)
- **%p** Es reemplazado por el equivalente de la localidad de las designaciones de AM/PM asociadas con un reloj de 12 horas
- **%S** Es reemplazado por el segundo como un número decimal (**00-61**)
- **%U** Es reemplazado por el número de la semana del año (el primer Domingo como el primer día de la semana 1) como un número decimal (**00-53**)
- **%w** Es reemplazado por el día de la semana como un número decimal (**0-6**), donde Domingo es **0**
- **%W** Es reemplazado por el número de la semana del año (el primer Lunes como el primer día de la semana 1) como un número decimal (**00-53**)
- **%x** Es reemplazado por la representación apropiada de la fecha de la localidad
- **%X** Es reemplazado por la representación apropiada de la hora de la localidad
- **%y** Es reemplazado por el año sin siglo como un número decimal (**00-99**)
- **%Y** Es reemplazado por el año con siglo como un número decimal
- **%Z** Es reemplazado por el nombre o la abreviatura del huso horario, o por ningunos caracteres si ningún huso horario es determinable

**%%** Es reemplazado por **%**

## Valor de retorno:

Si un especificador de conversión no es ninguno de los anteriores, el comprtamiento no está definido. Si el número total de los caracteres resultantes incluyendo el carácter nulo terminal no es mayor que **maxtam**, la función *strftime* retorna el número de caracteres colocados en el array apuntado por **cad** sin incluir el carácter nulo terminal. De lo contrario, cero es retornado y el conteido del array es indeterminado.

## Ejemplo:

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

int main()
{
  time_t tiempo;
  char cad[80];
  struct tm *tmPtr;

  tiempo = time(NULL);
  tmPtr = localtime(&tiempo);
  strftime( cad, 80, "%H:%M.%S, %A de %B de %Y", tmPtr );

  printf( "La hora local es: %s\n", asctime(tmPtr) );
  printf( "La hora y fecha locales son: %s\n", cad );

  return 0;
}
```

# Función strlen ANSI C

**Librería: string**

```
size_t strlen(const char *s);
```

Calcula el número de caracteres de la cadena apuntada por **s.**

## Valor de retorno:

La función retorna el número de caracteres hasta el carácter nulo, que no se incluye.

## Ejemplo:

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

int main()
{
   char s[13] = "Hola a todos";

   printf( "s=%s\n", s );
   printf( "strlen(s) = %d\n", strlen( s ) );

   return 0;
}
```

# Función strncat ANSI C

**Librería: string**

```
char *strncat(char*s1, const char *s2, size_t n);
```

Añade no más de **n** caracteres (un carácter nulo y los demás caracteres siguientes no son añadidos) de la cadena apuntada por **s2** al final de la cadena apuntada por **s1**. El carácter inicial de **s2** sobrescribe el carácter nulo al final de **s1**. El carácter nulo siempre es añadido al resultado.

## Valor de retorno:

La función retorna el valor de **s1**. Si la copia hace que los objetos se superpongan, entonces el comportamiento no está definido.

## Ejemplo:

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

int main()
{
   char s1[11] = "Hola ";
   char s2[6] = "amigos";

   printf( "s1=%s\t", s1 );
   printf( "s2=%s\n", s2 );
   strncat( s1, s2, 3 );
   printf( "s1=%s\n", s1 );

   return 0;
}
```

# Función strncmp ANSI C

**Librería: string**

```
int strncmp(const char *s1, const char *s2, size_t n);
```

Compara no más de **n** caracteres (caracteres posteriores al carácter nulo no se tienen en cuenta) de la cadena apuntada por **s1** con la cadena apuntada por **s2**.

## Valor de retorno:

La función retorna un número entero mayor, igual, o menor que cero, apropiadamente según la cadena apuntada por **s1** es mayor, igual, o menor que la cadena apuntada por **s2**.

## Ejemplo:

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

int main()
{
   char s1[9] = "artesano";
   char s2[8] = "artista";
   int i;

   printf( "s1=%s\t", s1 );
   printf( "s2=%s\n", s2 );

   i = strncmp( s1, s2, 3 );
   printf( "Las 3 primeras letras de s1 son " );
   if( i < 0 )  printf( "menores que" );
   else if( i > 0 )  printf( "mayores que" );
   else  printf( "iguales a" );
   printf( " s2\n" );

   return 0;
}
```

# Función strncpy ANSI C

**Librería: string**

```
char *strncpy(char *s1, const char *s2, size_t n);
```

Copia no más de **n** caracteres (caracteres posteriores al carácter nulo no son copiados) de la cadena apuntada por **s2** a la cadena apuntada por **s1**.

## Valor de retorno:

La función retorna el valor de **s1**. Si al copiar una cadena a la otra se superponen, entonces el comportamiento no está definido. Si el array/arreglo apuntado por **s2** es una cadena que es más corta que **n** caracteres, entonces caracteres nulos son añadidos a la copia en el array apuntado por **s1**.

## Ejemplo:

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

int main()
{
   char s2[8] = "abcdefg";
   char s1[8];

   strncpy( s1, s2, 3 );
   printf( "s2=%s\n", s2 );
   printf( "s1=%s\n", s1 );

   return 0;
}
```

# Función strpbrk ANSI C

**Librería: string**

```
char *strpbrk(const char *s1, const char *s2);
```

Localiza la primera aparición de la cadena apuntada por **s1** de cualquier carácter de la cadena apuntada por **s2**.

## Valor de retorno:

La función retorna un puntero al carácter, o un puntero nulo si ningún carácter de **s2** apareció en **s1**.

## Ejemplo:

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

int main()
{
   char s1[13] = "Hola a todos";
   char s2[5] = "deip";

   printf( "s1=%s\n", s1 );
   printf( "s2=%s\n", s2 );
   printf( "strpbrk(s1,s2) = %s\n", strpbrk( s1, s2 ) );

   return 0;
}
```

# Función strrchr ANSI C

**Librería: string**

```
char *strrchr(char *s, int c);
```

Localiza la última aparición de **c** (convertido a **unsigned char**) en la cadena apuntada por **s** (incluyendo el carácter nulo).

## Valor de retorno:

La función retorna un puntero a partir del carácter encontrado. Si no se ha encontrado el carácter, **c**, entonces retorna un puntero nulo.

## Ejemplo:

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

int main()
{
   char s[11] = "Hola amigos";
   char c = 'a';

   printf( "s=%s\t", s );
   printf( "c=%c\n", c );
   printf( "strrchr=%s\n", strrchr( s, c ) );

   return 0;
}
```

# Función strspn ANSI C

**Librería: string**

```
size_t strspn(const char *s1, const char *s2);
```

Calcula el número de caracteres de una subcadena inicial apuntada por **s1** que consiste en todos los caracteres formados en la cadena apuntada por **s2**.

## Valor de retorno:

La función retorna la longitud de esta subcadena.

## Ejemplo:

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

int main()
{
   char s1[13] = "Hola a todos";
   char s2[5] = "Hola";

   printf( "s1=%s\n", s1 );
   printf( "s2=%s\n", s2 );
   printf( "strspn(s1,s2) = %d\n", strspn( s1, s2 ) );

   return 0;
}
```

# Función strstr ANSI C

**Librería: string**

```
char *strstr(const char *s1, const char *s2);
```

Localiza la primera aparición en la cadena apuntada por **s1** de la secuencia de caracteres (excluyendo el carácter nulo) en la cadena apuntada por **s2**.

## Valor de retorno:

La función retorna un puntero a la cadena encontrada, o un puntero nulo si no se encontró la cadena. Si **s2** apunta a una cadena de longitud cero, la función retorna **s1**.

## Ejemplo:

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

int main()
{
   char s1[13] = "Hola a todos";
   char s2[3] = "la";

   printf( "s1=%s\n", s1 );
   printf( "s2=%s\n", s2 );
   printf( "strstr(s1,s2) = %s\n", strstr( s1, s2 ) );

   return 0;
}
```

# Función strtod ANSI C

**Librería: stdlib**

```
double strtod(const char *numPtr, char **finalPtr);
```

Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **long int**. Primero descompone la cadena de entrada en tres partes: una secuencia inicial, posiblemente vacía, de caracteres de espacio blanco (tal como es especificado por la macro isspace), una secuencia fuente asemejándose a una constante de coma flotante; y una cadena final de uno o más caracteres irreconocidos, incluyendo el carácter nulo final de la cadena entrada. Entonces, intenta convertir la secuencia fuente a un número de coma flotante, y retorna el resultado.

La forma expandida de la secuencia fuente es un signo, positivo o negativo, opcional, siguiendo una secuencia de dígitos opcionalmente conteniendo un carácter de la coma decimal (suele ser un punto), siguiendo un parte exponente opcional, pero sin incluir un sufijo de coma flotante. La secuencia fuente está definida como la subsecuencia inicial más larga de la cadena de entrada, comenzando por el primer carácter que no es un espacio blanco, que es de la forma esperada. La secuencia fuente no contiene caracteres si la cadena de entrada está vacía o consiste completamente de espacio en blanco, o si el primer carácter, que no es un espacio blanco, es distinto a un signo, un dígito, o un carácter de la coma decimal (o punto decimal).

Si la secuencia fuente tiene la forma esperada, la secuencia de caracteres comenzando por el primer dígito o el carácter de la coma decimal (cualquiera que aparezca primero) es interpretada como una constante de coma flotante, excepto que el carácter de la coma decimal es usado en lugar de la coma, y si no aparece la parte exponente ni el carácter de la coma flotante, la coma decimal es asumido que sigue el último dígito de la cadena. Si la secuencia fuente comienza con un signo negativo, el valor resultante de la conversión es negativo. Un puntero a la cadena final es guardado en el objeto apuntado por **finalPtr**, con tal de que **finalPtr** no es nulo.

Si la secuencia fuente está vacía o no tiene la forma esperada, ninguna conversión es realizada; el valor **numPtr** es guardado en el objeto apuntado por **finalPtr**, con tal de que **finalPtr** no es nulo.

A continuación, se muestra el formato usado por esta función:
**[eb] [sn] [ddd] [.] [ddd] [e[sn]ddd]**

donde:

| | |
|---|---|
| **[eb]** | Espacio Blanco opcional |
| **[sn]** | Signo opcional (+ ó -) |
| **e** | 'e' ó 'E' opcional |
| **[.]** | Coma decimal opcional (punto decimal) |
| **[ddd]** | Dígitos opcionales |

## Valor de retorno:

La función *strtod* retorna el valor convertido, si acaso existe. Si no se pudo realizar ninguna conversión, cero es retornado. Si el valor correcto no pertenece al intervalo de valores representables, HUGE_VAL positivo o negativo es retornado (según el signo del valor), y el valor de la macro **ERANGE** es guardado en errno. Si el valor correcto pudiera causar un "underflow", cero es retornado y el valor de la macro **ERANGE** es guardado en errno.

## Ejemplo:

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

int main()
```

```
{
   char *finalPtr;

   printf( "Convirtiendo la cadena \"%s\" en un numero: %e\n", "12.345678e-2",
strtod("12.345678e-2", &finalPtr) );
   printf( "Convirtiendo la cadena \"%s\" en un numero: %e\n", "-12.345678e+2",
strtod("-12.345678e+2", &finalPtr) );
   printf( "Convirtiendo la cadena \"%s\" en un numero: %e\n", "1.2345678",
strtod("1.2345678", &finalPtr) );
   printf( "Convirtiendo la cadena \"%s\" en un numero: %e\n", "1.2345678E-22",
strtod("1.2345678E-22", &finalPtr) );
   printf( "Convirtiendo la cadena \"%s\" en un numero: %e\n", "12345.678901234E14",
strtod("12345.678901234E14", &finalPtr) );

   return 0;
}
```

# Función strtok ANSI C

**Librería: string**

```
char *strtok(char *s1, const char *s2);
```

Rompe la cadena **s1** en segmentos o *tókens*. Esta ruptura destruye **s1**, en el proceso. La forma de romper la cadena depende de la secuencia de caracteres de la cadena **s2**. Estos caracteres se denominan *[caracteres] delimitadores*. La función recorrerá la cadena en busca de alguno de los delimitadores de la cadena **s2**. Cuando lo encuentre, el proceso se detiene, ya que tiene un token. Posteriores llamadas a strtok romperán la cadena **s1** en otros tókens. Estas llamadas pueden tener otra secuencia de delimitadores.

## Valor de retorno:

La primera llamada a strtok determina la cadena a romper, retornando el puntero al comienzo del primer token. Si se recorrió la cadena **s1** sin haber encontrado un delimitador, y aún no se ha obtenido el primer token, entonces la función retornará un puntero nulo.

Posteriores llamadas retornarán más tókens. Si ya no encuentra más delimitadores, entonces retornará todos los caracteres desde el último delimitador para ser el último token. Si ya se retornó el último token, entonces retornará un puntero nulo con demás llamadas a la función.

## Ejemplo:

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

int main()
{
   char s1[49] = "Esto es un ejemplo para usar la funcion strtok()";
   char s2[4] = " \n\t";
   char *ptr;

   printf( "s1=%s\n", s1 );

   ptr = strtok( s1, s2 );    // Primera llamada => Primer token
   printf( "%s\n", ptr );
   while( (ptr = strtok( NULL, s2 )) != NULL )    // Posteriores llamadas
     printf( "%s\n", ptr );

   return 0;
}
```

# Función strtol ANSI C

**Librería: stdlib**

```
long int strol(const char *numPtr, char **finalPtr, int base);
```

Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **long int**. Primero descompone la cadena de entrada en tres partes: una secuencia inicial, posiblemente vacía, de caracteres de espacio blanco (tal como es especificado por la función isspace), una secuencia fuente asemejándose a un entero representado en alguna base determinado por el valor de **base**, y una cadena final de uno o más caracteres irreconocidos, incluyendo el carácter nulo final de la cadena entrada. Entonces, intenta convertir la secuencia fuente a un entero, y retorna el resultado.

Si el valor de **base** es cero, la forma esperada de la secuencia fuente es aquélla de una constante entera, opcionalmente precedida por un signo más o menos, pero sin incluir un sufijo entero. Si el valor de **base** está entre 2 y 36, la forma esperada de la secuencia fuente es una secuencia de letras y dígitos representando un entero con una base especificado por **base**, opcionalmente precedida por un signo positivo o negativo, pero sin incluir un sufijo entero. Las letras de **a** (ó **A**) hasta **z** (ó **Z**) son atribuidos los valores de 10 a 35; sólo letras cuyos valores atribuidos son menores que aquéllos de la **base** están permitidos. Si el valor de **base** es 16, los caracteres **0x** ó **0X** puedes opcionalmente preceder la secuencia de letras y dígitos, a continuación del signo, si éste está presente.

La secuencia fuente está definida como la secuencia inicial más larga de la cadena de entrada, comenzando por el primer carácter que no es un espacio blanco, que es de la forma esperada. La secuencia fuente no contiene caracteres si la cadena de entrada está vacía o consiste completamente de espacio en blanco, o si el primer carácter que no es un espacio blanco es distinto a un signo o letra o dígito permitido.

Si la secuencia fuente tiene la forma esperada y el valor de **base** es cero, la secuencia de caracteres comenzando por el primer dígito es interpretada como una constante entera. Si la secuencia fuente tiene la forma esperada y el valor de **base** está entre 2 y 36, es usada como la base para la conversión, atribuyendo a cada letra su valor dado tal como descrito anteriormente. Si la secuencia fuente comienza con un signo negativo, el valor resultante de la conversión es negativo. Un puntero a la cadena final es guardado en el objeto apuntado por **finalPtr**, con tal de que **finalPtr** no es nulo.

Si la secuencia fuente está vacía o no tiene la forma esperada, ninguna conversión es realizada; el valor **numPtr** es guardado en el objeto apuntado por **finalPtr**, con tal de que **finalPtr** no es nulo.

A continuación, se muestra el formato usado por esta función:
**[eb] [sn] [0] [x] [ddd]**,

donde:

| | |
|---|---|
| **[eb]** | Espacio Blanco opcional |
| **[sn]** | Signo opcional (+ ó -) |
| **[0]** | Cero opcional (0) |
| **[x]** | 'x' ó 'X' opcional |
| **[ddd]** | Dígitos opcionales |

Si **base** es cero, los primeros caracteres de **numPtr** determinan la base:

| Primer carácter | Segundo carácter | Cadena interpretada como... |
|---|---|---|
| 0 | 1 a 7 | Octal |
| 0 | x ó X | Hexadecimal |
| 1 a 9 | - (0 a 9) | Decimal |

## Valor de retorno:

La función *strtol* retorna el valor convertido, si acaso existe. Si no se pudo realizar ninguna conversión, cero es retornado. Si el valor correcto no pertenece al intervalo de valores representables, LONG_MAX o LONG_MIN es retornado (según el signo del valor), y el valor de la macro ERANGE es guardado en errno.

## Ejemplo:

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

int main()
{
   char numPtr[9] = "12345678", *finalPtr;
   int base;

   for( base=2; base<=17; base++ )
      printf( "Convirtiendo la cadena \"%s\" en un numero en base %d: %u\n", numPtr,
base, strtol(numPtr, &finalPtr, base) );

   return 0;
}
```

# Función strtoul ANSI C

**Librería: stdlib**

```
unsigned long int strtoul(const char *numPtr, char **finalPtr, int base);
```

Convierte la porción inicial de la cadena apuntada por **numPtr** a una representación de **unsigned long int**. La función *strtoul* funciona identicamente a la función strtol.

## Valor de retorno:

La función *strtoul* retorna el valor convertido, si acaso existe. Si no se pudo realizar ninguna conversión, cero es retornado. Si el valor correcto no pertenece al intervalo de valores representables, UONG_MAX es retornado, y el valor de la macro ERANGE es guardado errno.

## Ejemplo:

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

int main()
{
   char numPtr[9] = "12345678", *finalPtr;
   int base;

   for( base=2; base<=17; base++ )
      printf( "Convirtiendo la cadena \"%s\" en un numero en base %d: %u\n", numPtr,
base, strtoul(numPtr, &finalPtr, base) );

   return 0;
}
```

# Función strxfrm ANSI C

**Librería: string**

```
size_t strxfrm(char *s1, const char *s2, size_t n);
```

Transforma la cadena apuntada por **s2** y coloca la cadena resultante en el array/arreglo apuntado por **s1**. La transformación es tal que, si la función strcmp es aplicada a las dos cadenas transformadas, el valor de retorno corresponderá a los valores de retorno de la función strcoll. No más de **n** caracteres son colocados en el array resultante apuntado por **s1**, incluyendo el carácter nulo. Si **n** es cero, **s1** puede ser un puntero nulo.

## Valor de retorno:

La función retorna la longitud del array transformado (sin incluir el carácter nulo). Si el valor es **n** o más, el contenido del array apuntado por **s1** es indeterminado. Si se copian objetos que son superpuestos, el comportamiento no está definido.

## Ejemplo:

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

int main()
{
   char s2[7] = "abcdefg";
   char s1[7];
   int i;

   i = strxfrm( s1, s2, 4 );
   printf( "s2=%s\n", s2 );
   printf( "s1=%s\tlonguitud=%d\n", s1, i );

   return 0;
}
```

# Función system ANSI C

**Librería: stdlib**

```
int system(const char *cadena);
```

Pasa la cadena apuntada por **cadena** al entorno local para ser ejecutada por el "procesador de comandos" - también denominado "intérprete de comandos" - de una forma definida según la implementación.Un puntero nulo puede ser usado para **cadena** para comprobar si existe un procesador de comandos.

## Valor de retorno:

Si el argumento es un puntero nulo, la función *system* retorna un valor distinto a cero sólo si el procesador de comandos está disponible. Si el argumento no es un puntero nulo, la función *system* retorna un valor definido según la implementación.

## Ejemplo:

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

/* Sólo funcionará si "dir" es aceptable por el sistema: MS-DOS, por ejemplo */

int main( void )
{
   puts( "La lista de ficheros en el directorio actual, segun el comando \"dir\":" );

   system( "dir" );

   return 0;
}
```

# Función tan ANSI C

**Librería: math**

```
double tan(double x);
```

Calcula la tangente de **x** (medido en radianes).

## Valor de retorno:

La función tan retorna la tangente, en radianes.

## Ejemplo:

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

int main()
{
   double x = 3.1416/3.0;

   printf( "tan( %f ) = %f\n", x, tan(x) );
   return 0;
}
```

# Función tanh ANSI C

**Librería: math**

```
double tanh(double x);
```

Calcula la tangente hiperbólica de **x**.

## Valor de retorno:

La función tanh retorna la tangente hiperbólica.

## Ejemplo:

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

int main()
{
   double x = 3.0;

   printf( "tanh( %f ) = %f\n", x, tanh(x) );
   return 0;
}
```

# Función time ANSI C

**Librería: time**

```
time_t time(time_t *tiempoPtr);
```

La función *time* determina el tiempo en formato condensado.

## Valor de retorno:

La función *time* retorna la mejor aproximación por la implementación del tiempo actual en formato condensado. Si el tiempo no está disponible, la función retorna el valor **(time_t)-1**. Si **tiempoPtr** no es un puntero nulo, el valor de retorno es también asignado al objeto al que apunta.

## Ejemplo:

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

int main( void )
{
   long int i=0;
   time_t comienzo, final;
   struct tm *tiempoComienzoPtr, *tiempoFinalPtr;

   comienzo = time( NULL );
   for( i=0; i<10000; i++ )   printf( "-" );
   final = time( NULL );

   printf( "Comienzo: %u s\n", comienzo );
   printf( "Final: %u s\n", final );
   printf( "Número de segundos transcurridos desde el comienzo del programa: %f s\n",
difftime(final, comienzo) );

   tiempoComienzoPtr = gmtime( &comienzo );
   tiempoFinalPtr = gmtime( &final );
   printf( "Comienzo: %s\n", asctime(tiempoComienzoPtr) );
   printf( "Final: %s\n", asctime(tiempoFinalPtr) );

   return 0;
}
```

# Función tmpfile ANSI C

**Librería: stdio**

```
FILE *tmpfile(void);
```

Crea un fichero binario temporal que será borrado automáticamente cuando sea cerrado o al terminar el programa. Si el programa termina abnormalmente, entonces el comportamiento de la función está definido según la implementación del compilador; puede ser que el fichero sea borrado o no. El fichero es abierto para añado con el modo "wb+".

## Valor de retorno:

La función retorna un puntero al fichero que es creado, si la operación fue realizada con éxito. Si el fichero no pudo ser creado, entonces la función retorna un puntero nulo.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   FILE *fTmp;

   fTmp = tmpfile();
   printf( "Un fichero temporal " );
   if( fTmp )   printf( "ha sido creado.\n" );
   else   printf( "no pudo ser creado.\n" );

   return 0;
}
```

# Función tmpnam ANSI C

**Librería: stdio**

```
char *tmpnam(char *s);
```

Genera una cadena de caracteres que es un nombre válido para ficheros y que no es igual al nombre de un fichero existente. La función *tmpnam* genera una cadena diferente cada vez que es llamada, hasta un máximo de TMP_MAX veces. Si la función es llamada más veces que TMP_MAX, entonces el comportamiento de la función está definido según la implementación del compilador.

## Valor de retorno:

Si el argumento es un puntero nulo, entonces la función *tmpnam* deja el resultado en un objeto estático interno y retorna un puntero a dicho objeto. Llamadas posteriores a la función pueden modificar el mismo objeto. Si el argumento no es un puntero nulo, entonces es asumido que apunta a un array/arreglo de al menos L_tmpnam caracteres; la función *tmpnam* escribe el resultado en el array/arreglo y retorna el argumento como su valor.

## Ejemplo:

```
#include <stdio.h>

int main()
{
   char sNombre[L_tmpnam];

   tmpnam( sNombre );
   printf( "Un nombre de fichero temporal: %s\n", sNombre );

   return 0;
}
```

# Función tolower ANSI C

**Librería: ctype**

```
int tolower(int ch);
```

Convierte un carácter, en un parámetro entero ch, a minúscula.

## Valor de retorno:

ch debe estar en el rango 0 a 255, y si está entre *A* y *Z* lo convierte a su equivalente en el rango *a* a *z*, el resto de los valores no son modificados. El valor de retorno es el valor convertido si ch era una mayúscula, o el valor original en caso contrario.

**Nota:** los caracteres en acentuados, o con diéresis, en mayúscula y la *Ñ* no sufren modificaciones.

## Ejemplo:

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

int main()
{
   char cadena[] = "ESTO ES UNA CADENA DE PRUEBA";
   int i;

   for(i = 0; cadena[i]; i++)
      cadena[i] = tolower(cadena[i]);

   printf("%s\n", cadena);
   return 0;
}
```

# Función toupper ANSI C

**Librería: ctype**

```
int toupper(int ch);
```

Convierte un carácter, en un parámetro entero ch, a mayúscula.

## Valor de retorno:

ch debe estar en el rango 0 a 255, y si está entre *a* y *z* lo convierte a su equivalente en el rango *A* a *Z*, el resto de los valores no son modificados. El valor de retorno es el valor convertido si ch era una minúscula, o el valor original en caso contrario.

**Nota:** los caracteres en acentuados, o con diéresis, en minúscula y la *ñ* no sufren modificaciones.

## Ejemplo:

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

int main()
{
   char cadena[] = "esto es una cadena de prueba";
   int i;

   for(i = 0; cadena[i]; i++)
      cadena[i] = toupper(cadena[i]);

   printf("%s\n", cadena);
   return 0;
}
```

# Función ungetc ANSI C

**Librería: stdio**

```
int ungetc(char c, FILE *stream);
```

Esta función envía el carácter especificado por **c** (convertido a un **unsigned char**) en el stream de entrada apuntado por **stream**. Los caracteres apilados serán retornados por lecturas posteriores en ese stream en orden inverso al que fueron apilados. Una llamada interventiva y correcta (con el stream apuntado por **stream**) a una función de posicionamiento de fichero (fseek, fsetpos, o rewind) descarta cualquier carácter apilado para el stream. El almacenamiento externo correspondiente al stream no varía.

Un carácter apilado es garantizado. Si la función *ungetc* es llamada demasiadas veces al mismo stream sin una lectura interventiva o una operación de posicionamiento de fichero en ese stream, entonces la operación falla. Si el valor de **c** es igual a la macro EOF, la operación falla y el stream de entrada no varía.

Una llamada exitosa a la función *ungetc* despeja el indicador de final de fichero para el stream. El valor del indicador de posición de ficheros para el stream después de leer o descartar todos los caracteres apilados será el mismo que era antes de que los caracteres fueran apilados. Para un stream de texto, el valor de su indicador de posición de ficheros después de una llamada exitosa a la función *ungetc* no es especificado hasta que todos los caracteres apilados sean leídos o descartados. Para un stream binario, el valor de su indicador de posición de ficheros es determinado por cada llamada exitosa a la función *ungetc*; si su valor era cero antes de la llamada, es indeterminado después de la llamada.

## Valor de retorno:

La función *ungetc* retorna el carácter apilado después de la conversión. Si la operación falla, entonces retorna EOF.

## Ejemplo:

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

int main( void )
{
   int i=0;
   char c;

   puts( "Escriba un numero entero seguido por un caracter:" );

   /* lee caracteres hasta uno que no sea un dígito o hasta un EOF */
   while( (c=getchar()) != EOF && isdigit(c) )
      i = 10 * i + c - 48;    /* convierte ASCII un valor entero (int) */

   /* Si se leyó un carácter que no era un dígito, apílalo en el stream de entrada */
   if( c != EOF )
      ungetc( c, stdin );

   printf( "i = %d, siguiente caracter en el stream = %c\n", i, getchar() );

   return 0;
}
```

# Función vfprintf ANSI C

**Librería: stdio**

```
int vfprintf(FILE *stream, const char *formato, va_list argumentos);
```

Esta función es equivalente a fprintf, con la lista de argumentos de variables reemplazado por **argumentos**, que es inicializado por la macro va_start (y posiblemente por llamadas posteriores a va_arg. La función *vfprintf* no invoca la macro va_end.

## Valor de retorno:

La función *vfprintf* retorna el número de caracteres transmitidos, o un valor negativo si se produce un error de salida.

## Ejemplo:

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

int mi_vfprintf( FILE *stream, const char *formato, ... )
{
   va_list listaPtr;
   int resultado=0;

   va_start( listaPtr, formato );
   resultado = vfprintf( stream, formato, listaPtr );
   va_end( listaPtr );

   return resultado;
}

int main()
{
   FILE *fichero;
   char nombre[11] = "datos6.dat";
   unsigned int i;

   fichero = fopen( nombre, "w" );
   printf( "Fichero: %s -> ", nombre );
   if( fichero )
      printf( "creado (ABIERTO)\n" );
   else
   {
      printf( "Error (NO ABIERTO)\n" );
      return 1;
   }

   mi_vfprintf( fichero, "Esto es un ejemplo de usar la funcion \'vfprintf\'\n" );
   mi_vfprintf( fichero, "\t 2\t 3\t 4\n" );
   mi_vfprintf( fichero, "x\tx\tx\tx\n\n" );
   for( i=1; i<=10; i++ )
      mi_vfprintf( fichero, "%d\t%d\t%d\t%d\n", i, i*i, i*i*i, i*i*i*i );

   mi_vfprintf( stdout, "Datos guardados en el fichero: %s\n", nombre );
   if( !fclose(fichero) )
      printf( "Fichero cerrado\n" );
```

```
   else
   {
      printf( "Error: fichero NO CERRADO\n" );
      return 1;
   }

   return 0;
}
```

# Función vprintf ANSI C

**Librería: stdio**

```
int vprintf(const char *formato, va_list argumentos);
```

Esta función es equivalente a printf, con la lista de argumentos de variables reemplazado por **argumentos**, que habrá sido inicializado por la macro va_start (y posiblemente por llamadas posteriores a va_arg. La función *vprintf* no invoca la macro va_end.

## Valor de retorno:

La función *vprintf* retorna el número de caracteres transmitidos, o un valor negativo si se produce un error de salida.

## Ejemplo:

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

int mi_vprintf( const char *formato, ... )
{
   va_list listaPtr;
   int resultado=0;

   va_start( listaPtr, formato );
   resultado = vprintf( formato, listaPtr );
   va_end( listaPtr );

   return resultado;
}

int main()
{
   char nombre[20];
   unsigned int edad=0;

   mi_vprintf( "Escriba su nombre: " );
   scanf( "%s", nombre );
   mi_vprintf( "Escriba su edad: " );
   scanf( "%u", &edad );
```

```
    mi_vprintf( "\nHola %s. Tienes %d anyos.\n", nombre, edad );

    return 0;
}
```

# Función vsprintf ANSI C

**Librería: stdio**

```
int vsprintf(char *cadena, const char *formato, va_list argumentos);
```

Esta función es equivalente a sprintf, con la lista de argumentos de variables reemplazado por **argumentos**, que habrá sido inicializado por la macro va_start (y posiblemente por llamadas posteriores a va_arg. La función *vsprintf* no invoca la macro va_end. Si se copian objetos que se superponen, entonces el comportamiento no está definido.

## Valor de retorno:

La función *vsprintf* retorna el número de caracteres escritos al array, sin contar el carácter nulo al final.

## Ejemplo:

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

int mi_vsprintf( char *cadena, const char *formato, ... )
{
   va_list listaPtr;
   int resultado=0;

   va_start( listaPtr, formato );
   resultado = vsprintf( cadena, formato, listaPtr );
   va_end( listaPtr );

   return resultado;
}

int main()
{
   char nombre[20], mensaje[81];
   unsigned int edad=0;

   printf( "Escriba su nombre: " );
   scanf( "%s", nombre );
   printf( "Escriba su edad: " );
   scanf( "%u", &edad );

   mi_vsprintf( mensaje, "\nHola %s. Tienes %d anyos.\n", nombre, edad );
   puts( mensaje );

   return 0;
}
```

# Función wctomb ANSI C

**Librería: stdlib**

```
int wctomb(char *cad, wchar_t wchar);
```

Determina el número de bytes necesitado para representar un carácter multibyte correspondiendo al código cuyo valor es **wchar** (incluyendo cualquier cambio en el estado de traslado). Guard la representación del carácter multibyte en el array apuntado por **cad** (si **cad** no es un puntero nulo). Al menos MB_CUR_MAX caracteres son guardados. Si el valor de **wchar** es cero, la función *wctomb* es dejado en el estado inical de traslado.

## Valor de retorno:

Si **cad** es un puntero nulo, la función *wctomb* retorna un valor distinto a cero o cero, si los códigos del carácter multibyte, respectivamente, tienen o no tienen códigos dependiendo del estado. Si **cad** no es un puntero nulo, la función *wctomb* retorna **-1** si el valor de **wchar** no corresponde a un carácter multibyte válido, o retorna el número de bytes que están contenidos en el carácter multibyte correspondiendo al valor de **wchar**. En ningún caso, el valor retornado será mayor que **n** o el valor de la macro MB_CUR_MAX.

## Ejemplo:

```
/* Ejemplo sacado de la ayuda de Borland */

#include <stdlib.h>
#include <stdio.h>

int main()
{
  int     x;
  char     *mbchar    = (char *)calloc(1, sizeof( char));
  wchar_t  wchar      = L'a';
  wchar_t  *pwcnull   = NULL;
  wchar_t  *pwchar    = (wchar_t *)calloc(1,  sizeof( wchar_t ));

  printf( "Convertir un carácter ancho a un carácter multibyte:\n" );
  x = wctomb( mbchar, wchar);
  printf( "\tCaracteres convertidos: %u\n", x );
  printf( "\tCarácter multibyte: %x\n\n", mbchar );

  printf( "Tamaño del carácter multibyte (según mblen): %u\n", mblen(mbchar,
MB_CUR_MAX) );
  printf( "Convertir carácter multibyte de nuevo a un carácter ancho:\n" );

  x = mbtowc( pwchar, mbchar, MB_CUR_MAX );
  printf( "\tBytes convertidos: %u\n", x );
  printf( "\tCarácter ancho: %x\n\n", pwchar );

  printf( "Intentar convertir cuando el destinatario es nulo (NULL)\n" );
  printf( " retorna la longitud del carácter multibyte: " );
  x = mbtowc( pwcnull, mbchar, MB_CUR_MAX );
```

```
  printf( "%u\n\n", x );

  printf( "Intenta convertir un puntero nulo (NULL) a un" );
  printf( " carácter ancho:\n" );
  mbchar = NULL;
  x = mbtowc( pwchar, mbchar, MB_CUR_MAX );

  printf( "\tBytes convertidos: %u\n", x );

  return 0;
}
```

# Indice de macros

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## -A-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| assert | assert | assert.h | cassert |

## -B-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| BUFSIZ | stdio | stdio.h | cstdio |

## -C-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| CLOCKS_PER_SEC | time | time.h | ctime |

## -D-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| DBL_DIG | float | float.h | cfloat |
| DBL_EPSILON | float | float.h | cfloat |
| DBL_MANT_DIG | float | float.h | cfloat |
| DBL_MAX | float | float.h | cfloat |
| DBL_MAX_10_EXP | float | float.h | cfloat |
| DBL_MAX_EXP | float | float.h | cfloat |
| DBL_MIN | float | float.h | cfloat |
| DBL_MIN_10_EXP | float | float.h | cfloat |
| DBL_MIN_EXP | float | float.h | cfloat |

## -E-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| EOF | stdio | stdio.h | cstdio |
| errno | errno | errno.h | cerrno |
| EXIT_FAILURE | stdlib | stdlib.h | cstdlib |
| EXIT_SUCCESS | stdlib | stdlib.h | cstdlib |

## -F-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| FILENAME_MAX | stdio | stdio.h | cstdio |
| FLT_DIG | float | float.h | cfloat |
| FLT_EPSILON | float | float.h | cfloat |
| FLT_MANT_DIG | float | float.h | cfloat |
| FLT_MAX | float | float.h | cfloat |
| FLT_MAX_10_EXP | float | float.h | cfloat |
| FLT_MAX_EXP | float | float.h | cfloat |
| FLT_MIN | float | float.h | cfloat |
| FLT_MIN_10_EXP | float | float.h | cfloat |
| FLT_MIN_EXP | float | float.h | cfloat |
| FLT_RADIX | float | float.h | cfloat |
| FLT_ROUNDS | float | float.h | cfloat |
| FOPEN_MAX | stdio | stdio.h | cstdio |

## -H-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| HUGE_VAL | math | math.h | cmath |

## -I-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| isalnum | ctype | ctype.h | cctype |
| isalpha | ctype | ctype.h | cctype |
| isascii | ctype | ctype.h | cctype |
| iscntrl | ctype | ctype.h | cctype |
| isdigit | ctype | ctype.h | cctype |
| isgraph | ctype | ctype.h | cctype |
| islower | ctype | ctype.h | cctype |
| isprint | ctype | ctype.h | cctype |
| ispunct | ctype | ctype.h | cctype |
| isspace | ctype | ctype.h | cctype |
| isupper | ctype | ctype.h | cctype |
| isxdigit | ctype | ctype.h | cctype |

## -L-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| LC_ALL | locale | locale.h | clocale |
| LC_COLLATE | locale | locale.h | clocale |
| LC_CTYPE | locale | locale.h | clocale |
| LC_MONETARY | locale | locale.h | clocale |
| LC_NUMERIC | locale | locale.h | clocale |
| LC_TIME | locale | locale.h | clocale |
| LDBL_DIG | float | float.h | cfloat |
| LDBL_EPSILON | float | float.h | cfloat |
| LDBL_MANT_DIG | float | float.h | cfloat |
| LDBL_MAX | float | float.h | cfloat |
| LDBL_MAX_10_EXP | float | float.h | cfloat |
| LDBL_MAX_EXP | float | float.h | cfloat |
| LDBL_MIN | float | float.h | cfloat |
| LDBL_MIN_10_EXP | float | float.h | cfloat |

| | | | |
|---|---|---|---|
| LDBL_MIN_EXP | float | float.h | cfloat |
| L_tmpnam | stdio | stdio.h | cstdio |

## -M-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| MB_CUR_MAX | stdlib | stdlib.h | cstdlib |

## -N-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| NULL | locale | locale.h | clocale |
| NULL | stddef | stddef.h | cstddef |
| NULL | stdio | stdio.h | cstdio |
| NULL | stdlib | stdlib.h | cstdlib |
| NULL | string | string.h | cstring |
| NULL | time | time.h | ctime |

## -O-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| offsetof | stddef | stddef.h | cstddef |

## -R-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| RAND_MAX | stdlib | stdlib.h | cstdlib |

## -S-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| SEEK_CUR | stdio | stdio.h | cstdio |

| | | | |
|---|---|---|---|
| SEEK_END | stdio | stdio.h | cstdio |
| SEEK_SET | stdio | stdio.h | cstdio |
| SIGABRT | signal | signal.h | csignal |
| SIGFPE | signal | signal.h | csignal |
| SIGILL | signal | signal.h | csignal |
| SIGSEGV | signal | signal.h | csignal |
| SIGTERM | signal | signal.h | csignal |
| SIG_DFL | signal | signal.h | csignal |
| SIG_ERR | signal | signal.h | csignal |
| SIG_IGN | signal | signal.h | csignal |
| stderr | stdio | stdio.h | cstdio |
| stdin | stdio | stdio.h | cstdio |
| stdout | stdio | stdio.h | cstdio |

## -T-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| TMP_MAX | stdio | stdio.h | cstdio |
| toascii | ctype | ctype.h | cctype |

## -V-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| va_arg | stdarg | stdarg.h | cstdarg |
| va_end | stdarg | stdarg.h | cstdarg |
| va_list | stdarg | stdarg.h | cstdarg |
| va_start | stdarg | stdarg.h | cstdarg |

## -_-

| Macro | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| _IOFBF | stdio | stdio.h | cstdio |
| _IOLBF | stdio | stdio.h | cstdio |

| | | | |
|---|---|---|---|
| _IONBF | stdio | stdio.h | cstdio |

# Macro assert ANSI C

**Librería: assert**

```
void assert(int test);
```

Comprueba la condición 'test' y dependiendo del resultado, puede abortar el programa.

Se trata de una macro que se expande como una sentencia "if", si 'test' se evalua como cero, la función aborta el programa y muestra el siguiente mensaje en stderr:

Assertion failed: <condición>, fichero <nombre de fichero>, line <número de línea>

El nombre de fichero y el número de línea corresponden con el fichero y línea en la que está la macro.

Si se coloca la directiva "#define NDEBUG"; (no depurar) en el fichero fuente antes de la directiva "#include <assert.h>", todas las sentencias assert serán comentadas.

# Macro BUFSIZ ANSI C

**Librería: stdio**

```
#define BUFSIZ
```

Una expresión entera constante, el cual es el tamaño usado por la función setbuf.

## Ejemplo:

| | |
|---|---|
| #define BUFSIZ 16384 | **DJGPP** |
| #define BUFSIZ 512 | **Borland** |
| #define BUFSIZ 512 | **Dev-C++** |

# Macro CLOCKS_PER_SEC ANSI C

**Librería: time**

```
#define CLOCKS_PER_SEC
```

El número por segundos del valor retornado por la función clock.

## Ejemplo:

```
#define CLOCKS_PER_SEC 91
```

**DJGPP**

```
#define CLOCKS_PER_SEC 1000.0
```

**Borland**

```
#define CLOCKS_PER_SEC  1000.0
```

**Dev-C++**

# Macros FLT_DIG, DBL_DIG, LDBL_DIG ANSI C

**Librería: float**

```
#define FLT_DIG      6
#define DBL_DIG      10
#define LDBL_DIG     10
```

El número de dígitos decimales, *q*, tal que cualquier número de coma flotante con *q* dígitos decimales puede ser redondeado en un número de coma flotante con base *p* y *b* dígitos y de vuelta otra vez sin alterar los *q* dígitos decimales.

## Ejemplo:

```
#define FLT_DIG          6
#define DBL_DIG          15
#define LDBL_DIG         18
```

**DJGPP**

```
#define FLT_DIG          6
#define DBL_DIG          15
#define LDBL_DIG         18
```

**Borland**

```
#define FLT_DIG          6
#define DBL_DIG          15
#define LDBL_DIG         15
```

**Dev-C++**

# Macros FLT_EPSILON, DBL_EPSILON, LDBL_EPSILON ANSI C

**Librería: float**

```
#define FLT_EPSILON    1E-5
#define DBL_EPSILON    1E-9
#define LDBL_EPSILON   1E-9
```

La diferencia entre 1.0 y el menor valor que es mayor que 1.0 que es representable en el tipo de coma flotante dado.

## Ejemplo:

**DJGPP**

```
extern float FLT_EPSILON;
extern double DBL_EPSILON;
extern long double LDBL_EPSILON;
```

**Borland**

```
#define FLT_EPSILON             1.19209290E-07F
#define DBL_EPSILON             2.2204460492503131E-16
#define LDBL_EPSILON            1.08420217248550443 4e-019L
```

**Dev-C++**

```
#define FLT_EPSILON         1.19209290e-07F
#define DBL_EPSILON         1.1102230246251568e-16
#define LDBL_EPSILON        1.1102230246251568e-16L
```

# Macros FLT_MANT_DIG, DBL_MANT_DIG, LDBL_MANT_DIG ANSI C

**Librería: float**

```
#define FLT_MANT_DIG
#define DBL_MANT_DIG
#define LDBL_MANT_DIG
```

El número de dígitos de base FLT_RADIX en la significante de coma flotante, *p*.

## Ejemplo:

```
#define FLT_MANT_DIG          24
#define DBL_MANT_DIG          53
#define LDBL_MANT_DIG         64
```

**DJGPP**

```
#define FLT_MANT_DIG          24
#define DBL_MANT_DIG          53
#define LDBL_MANT_DIG         64
```

**Borland**

```
#define FLT_MANT_DIG          24
#define DBL_MANT_DIG          53
#define LDBL_MANT_DIG         53
```

**Dev-C++**

# Macros FLT_MAX, DBL_MAX, LDBL_MAX ANSI C

**Librería: float**

```
#define FLT_MAX    1E+37
#define DBL_MAX    1E+37
#define LDBL_MAX   1E+37
```

El mayor número finito y representable de coma flotante.

## Ejemplo:

**DJGPP**

```
extern float __dj_float_max;
extern double __dj_double_max;
extern long double __dj_long_double_max;

#define FLT_MAX    __dj_float_max
#define DBL_MAX    __dj_double_max
#define LDBL_MAX   __dj_long_double_max
```

**Borland**

```
extern float        _RTLENTRY _EXPDATA _huge_flt;
extern double       _RTLENTRY _EXPDATA _huge_dble;
extern long double  _RTLENTRY _EXPDATA _huge_ldble;

#define FLT_MAX             _huge_flt
#define DBL_MAX             _huge_dble
#define LDBL_MAX            _huge_ldble
```

**Dev-C++**

```
#define FLT_MAX         3.40282347e+38F
#define DBL_MAX         1.7976931348623157e+308
#define LDBL_MAX        1.7976931348623157e+308L
```

# Macros FLT_MAX_10_EXP, DBL_MAX_10_EXP, LDBL_MAX_10_EXP ANSI C

**Librería: float**

```
#define FLT_MAX_10_EXP     +37
#define DBL_MAX_10_EXP    +37
#define LDBL_MAX_10_EXP  +37
```

El mayor número entero tal que 10 elevado a la potencia de éste está dentro del alcance de números finitos y representables de coma flotante.

## Ejemplo:

```
#define FLT_MAX_10_EXP      38
#define DBL_MAX_10_EXP      308
#define LDBL_MAX_10_EXP     308
```

**DJGPP**

```
#define FLT_MAX_10_EXP          +38
#define DBL_MAX_10_EXP          +308
#define LDBL_MAX_10_EXP         +4932
```

**Borland**

```
#define FLT_MAX_10_EXP      38
#define DBL_MAX_10_EXP      308
#define LDBL_MAX_10_EXP     308
```

**Dev-C++**

# Macros FLT_MAX_EXP, DBL_MAX_EXP, LDBL_MAX_EXP ANSI C

**Librería: float**

```
#define FLT_MAX_EXP
#define DBL_MAX_EXP
#define LDBL_MAX_EXP
```

El mayor número entero tal que FLT_RADIX elevado a la potencia de éste menos 1 es un número finito y representable de coma flotante.

## Ejemplo:

```
#define FLT_MAX_EXP     128
#define DBL_MAX_EXP     1024
#define LDBL_MAX_EXP    16384
```

**DJGPP**

```
#define FLT_MAX_EXP     +128
#define DBL_MAX_EXP     +1024
#define LDBL_MAX_EXP    +16384
```

**Borland**

```
#define FLT_MAX_EXP     128
#define DBL_MAX_EXP     1024
#define LDBL_MAX_EXP    1024
```

**Dev-C++**

# Macros FLT_MIN, DBL_MIN, LDBL_MIN ANSI C

**Librería: float**

```
#define FLT_MIN    1E-37
#define DBL_MIN    1E-37
#define LDBL_MIN   1E-37
```

El menor número normalizado y positivo de coma flotante.

## Ejemplo:

**DJGPP**

```
extern float __dj_float_min;
extern double __dj_double_min;
extern long double __dj_long_double_min;

#define FLT_MIN    __dj_float_min
#define DBL_MIN    __dj_double_min
#define LDBL_MIN   __dj_long_double_min
```

**Borland**

```
extern long double  _RTLENTRY _EXPDATA _tiny_ldble;

#define FLT_MAX              1.17549435E-38F
#define DBL_MAX              2.2250738585072014E-308
#define LDBL_MAX             _tiny_ldble
```

**Dev-C++**

```
#define FLT_MAX         1.17549435e-38F
#define DBL_MAX         2.2250738585072014e-308
#define LDBL_MAX        2.2250738585072014e-308L
```

# Macros FLT_MIN_10_EXP, DBL_MIN_10_EXP, LDBL_MIN_10_EXP ANSI C

**Librería: float**

```
#define FLT_MIN_10_EXP    -37
#define DBL_MIN_10_EXP    -37
#define LDBL_MIN_10_EXP   -37
```

El menor número negativo tal que 10 elevado a la potencia de éste está dentro del alcance de números de coma flotante normalizados.

## Ejemplo:

```
#define FLT_MIN_10_EXP     (-37)
#define DBL_MIN_10_EXP     (-307)
#define LDBL_MIN_10_EXP    (-4931)
```

**DJGPP**

```
#define FLT_MIN_10_EXP      -37
#define DBL_MIN_10_EXP      -307
#define LDBL_MIN_10_EXP     -4931
```

**Borland**

```
#define FLT_MIN_10_EXP     (-37)
#define DBL_MIN_10_EXP     (-307)
#define LDBL_MIN_10_EXP    (-307)
```

**Dev-C++**

# Macros FLT_MIN_EXP, DBL_MIN_EXP, LDBL_MIN_EXP ANSI C

**Librería: float**

```
#define FLT_MIN_EXP
#define DBL_MIN_EXP
#define LDBL_MIN_EXP
```

El menor número entero negativo tal que FLT_RADIX elevado a la potencia de éste menos 1 es un número normalizado de coma flotante.

## Ejemplo:

```
#define FLT_MIN_EXP     (-125)
#define DBL_MIN_EXP     (-1021)
#define LDBL_MIN_EXP    (-16381)
```

**DJGPP**

```
#define FLT_MIN_EXP      -125
#define DBL_MIN_EXP      -1021
#define LDBL_MIN_EXP     -16381
```

**Borland**

```
#define FLT_MIN_EXP     (-125)
#define DBL_MIN_EXP     (-1021)
#define LDBL_MIN_EXP    (-1021)
```

**Dev-C++**

# Macro EOF ANSI C

**Librería: stdio**

```
#define EOF
```

Una expresión entera constante y negativa que es retornada por varias funciones para indicar un final de fichero (End Of File), esto es, no hay más datos de entrada de un stream.

## Ejemplo:

| | |
|---|---|
| #define EOF (-1) | **DJGPP** |
| #define EOF (-1) | **Borland** |
| #define EOF (-1) | **Dev-C++** |

# Definiciones de errno ANSI C

**Librería: errno**

La definición de estas constantes puede depender del compilador, y se incluyen aquí, sólo como ejemplo.

En el caso de Dev-C++, estas constantes son:

```
#define EPERM           1       /* Operation not permitted */
#define ENOFILE         2       /* No such file or directory */
#define ENOENT          2
#define ESRCH           3       /* No such process */
#define EINTR           4       /* Interrupted function call */
#define EIO             5       /* Input/output error */
#define ENXIO           6       /* No such device or address */
#define E2BIG           7       /* Arg list too long */
#define ENOEXEC         8       /* Exec format error */
#define EBADF           9       /* Bad file descriptor */
#define ECHILD          10      /* No child processes */
#define EAGAIN          11      /* Resource temporarily unavailable */
#define ENOMEM          12      /* Not enough space */
#define EACCES          13      /* Permission denied */
#define EFAULT          14      /* Bad address */
/* 15 - Unknown Error */
#define EBUSY           16      /* strerror reports "Resource device" */
#define EEXIST          17      /* File exists */
#define EXDEV           18      /* Improper link (cross-device link?) */
#define ENODEV          19      /* No such device */
#define ENOTDIR         20      /* Not a directory */
#define EISDIR          21      /* Is a directory */
#define EINVAL          22      /* Invalid argument */
#define ENFILE          23      /* Too many open files in system */
#define EMFILE          24      /* Too many open files */
#define ENOTTY          25      /* Inappropriate I/O control operation */
/* 26 - Unknown Error */
#define EFBIG           27      /* File too large */
#define ENOSPC          28      /* No space left on device */
#define ESPIPE          29      /* Invalid seek (seek on a pipe?) */
#define EROFS           30      /* Read-only file system */
#define EMLINK          31      /* Too many links */
#define EPIPE           32      /* Broken pipe */
#define EDOM            33      /* Domain error (math functions) */
#define ERANGE          34      /* Result too large (possibly too small) */
/* 35 - Unknown Error */
#define EDEADLOCK       36      /* Resource deadlock avoided (non-Cyg) */
#define EDEADLK         36
/* 37 - Unknown Error */
#define ENAMETOOLONG    38      /* Filename too long (91 in Cyg?) */
#define ENOLCK          39      /* No locks available (46 in Cyg?) */
#define ENOSYS          40      /* Function not implemented (88 in Cyg?) */
#define ENOTEMPTY       41      /* Directory not empty (90 in Cyg?) */
#define EILSEQ          42      /* Illegal byte sequence */
```

En el caso de Borland C++, son:

```
#define EZERO    0      /* Error 0                    */
#define EINVFNC  1      /* Invalid function number    */
#define ENOFILE  2      /* File not found             */
#define ENOPATH  3      /* Path not found             */
#define ECONTR   7      /* Memory blocks destroyed    */
#define EINVMEM  9      /* Invalid memory block address */
#define EINVENV 10      /* Invalid environment        */
#define EINVFMT 11      /* Invalid format             */
#define EINVACC 12      /* Invalid access code        */
#define EINVDAT 13      /* Invalid data               */
#define EINVDRV 15      /* Invalid drive specified    */
#define ECURDIR 16      /* Attempt to remove CurDir   */
#define ENOTSAM 17      /* Not same device            */
#define ENMFILE 18      /* No more files              */

#define ENOENT   2      /* No such file or directory*/
#define EMFILE   4      /* Too many open files        */
#define EACCES   5      /* Permission denied          */
#define EBADF    6      /* Bad file number            */
#define ENOMEM   8      /* Not enough core            */
#define EFAULT  14      /* Unknown error              */
#define ENODEV  15      /* No such device             */
#define EINVAL  19      /* Invalid argument           */
#define E2BIG   20      /* Arg list too long          */
#define ENOEXEC 21      /* Exec format error          */
#define EXDEV   22      /* Cross-device link          */
#define ENFILE  23      /* Too many open files        */
#define ECHILD  24      /* No child process           */
#define ENOTTY  25      /* UNIX - not MSDOS           */
#define ETXTBSY 26      /* UNIX - not MSDOS           */
#define EFBIG   27      /* UNIX - not MSDOS           */
#define ENOSPC  28      /* No space left on device    */
#define ESPIPE  29      /* Illegal seek               */
#define EROFS   30      /* Read-only file system      */
#define EMLINK  31      /* UNIX - not MSDOS           */
#define EPIPE   32      /* Broken pipe                */
#define EDOM    33      /* Math argument              */
#define ERANGE  34      /* Result too large           */
#define EEXIST  35      /* File already exists        */
#define EDEADLOCK 36    /* Locking violation          */
#define EPERM   37      /* Operation not permitted    */
#define ESRCH   38      /* UNIX - not MSDOS           */
#define EINTR   39      /* Interrupted function call */
#define EIO     40      /* Input/output error         */
#define ENXIO   41      /* No such device or address */
#define EAGAIN  42      /* Resource temporarily unavailable */
#define ENOTBLK 43      /* UNIX - not MSDOS           */
#define EBUSY   44      /* Resource busy              */
#define ENOTDIR 45      /* UNIX - not MSDOS           */
#define EISDIR  46      /* UNIX - not MSDOS           */
#define EUCLEAN 47      /* UNIX - not MSDOS           */
```

También se declara la variable:

```
extern int errno;
```

Esta variable se usa por la función perror de stdio.h, para obtener los mensajes de error.

# Macros EXIT_FAILURE, EXIT_SUCCESS ANSI C

**Librería: stdlib**

```
#define EXIT_FAILURE
#define EXIT_SUCCESS
```

Expresiones enteras que pueden ser usadas como el argumento para la función exit para retornar el estado de terminación sin o con éxito, respectivamente, al entorno local.

## Ejemplo:

```
#define EXIT_FAILURE 1
#define EXIT_SUCCESS 0
```

**DJGPP**

```
#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1
```

**Borland**

```
#define EXIT_SUCCESS   0
#define EXIT_FAILURE   -1
```

**Dev-C++**

# Macro FILENAME_MAX ANSI C

**Librería: stdio**

```
#define FILENAME_MAX
```

Una expresión constante entera que es el tamaño necesitado para el array de tipo **char** suficientemente grande para guardar la cadena del nombre de fichero más larga que la implementación garantiza pueda ser abierta.

## Ejemplo:

| | |
|---|---|
| #define FILENAME_MAX 260 | **DJGPP** |
| #define FILENAME_MAX 80 | **Borland** |
| #define FILENAME_MAX (260) | **Dev-C++** |

# Macro FLT_RADIX ANSI C

**Librería: float**

```
#define FLT_RADIX       2
```

La base de la representación del exponente, *b*.

## Ejemplo:

```
#define FLT_RADIX       2
```

**DJGPP**

```
#define FLT_RADIX       2
```

**Borland**

```
#define FLT_RADIX       2
```

**Dev-C++**

# Macro FLT_ROUNDS ANSI C

**Librería: float**

```
#define FLT_ROUNDS
```

El modo para redondear para la suma de números de coma flotante.

- **-1** Indeterminado
- **0** Hacia cero
- **1** Hacia el más cercano
- **2** Hacia el infinito positivo
- **3** Hacia el infinito negativo

## Ejemplo:

```
#define FLT_ROUNDS      1
```

**DJGPP**

```
#define FLT_ROUNDS      1
```

**Borland**

```
#define FLT_ROUNDS      1
```

**Dev-C++**

# Macro FOPEN_MAX ANSI C

**Librería: stdio**

```
#define FOPEN_MAX
```

Una expresión constante entera que es el número mínimo de ficheros la implementación garantiza pueden ser abiertos simultáneamente.

## Ejemplo:

| | |
|---|---|
| #define FOPEN_MAX 20 | **DJGPP** |
| #define _NFILE_ 50<br>#define FOPEN_MAX (_NFILE_ - 2) /* STDC */<br>#define FOPEN_MAX _NFILE_ /* Normal */ | **Borland** |
| #define FOPEN_MAX (20) | **Dev-C++** |

# Constante HUGE_VAL ANSI C

**Librería: math**

```
#define HUGE_VAL
```

Una constante simbólica representando una expresión positiva de tipo **double**.

# Macro isalnum ANSI C

**Librería: ctype**

```
int isalnum(int c);
```

Comprueba sin un carácter es alfanumérico.

isalnum es una macro que verifica el entero c pertenece al rango de letras (*A* a *Z* o *a* a *z*) o al de dígitos (*0* a *9*), por defecto. La verificación se hace mediante una tabla, y su comportamiento depende le la categoría LC_CTYPE actual.

## Valor de retorno:

El valor de retorno será no nulo si c es una letra o un número, y cero en caso contrario.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0sR(h&R1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isalnum(cadena[i]));

   return 0;
}
```

# Macro isalpha ANSI C

**Librería: ctype**

```
int isalpha(int c);
```

Comprueba sin un carácter es alfabético.

isalpha es una macro que verifica el entero c pertenece al rango de letras (*A* a *Z* o *a* a *z*), por defecto. La verificación se hace mediante una tabla, y su comportamiento depende le la categoría LC_CTYPE actual.

## Valor de retorno:

El valor de retorno será no nulo si c es una letra y cero en caso contrario.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0sR(h&R1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isalpha(cadena[i]));

   return 0;
}
```

# Macro isascii ANSI C

**Librería: ctype**

```
int isascii(int c);
```

Comprueba sin un carácter pertenece al ASCII de 7 bits.

isascii es una macro que verifica el entero c pertenece al rango de (*0* a *127*). Esta macro está definida para todos lo valores enteros.

## Valor de retorno:

El valor de retorno será no nulo si c está en el rango entre 0 y 127, en hezadecimal entre 0x00 y 0x7f.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0ñsáR(h&~?RÛ1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isascii(cadena[i]));

   return 0;
}
```

# Macro iscntrl ANSI C

**Librería: ctype**

```
int iscntrl(int c);
```

Comprueba sin un carácter es de control.

iscntrl es una macro que verifica el entero c pertenece al rango caracteres de control, que depende de la categoría local LC_CTYPE, por defecto, el rango es de (*0x00* a *0x1F* y *0x7F*).

## Valor de retorno:

El valor de retorno será no nulo si c es un carácter "delete" o un carácter de control.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0ñs\003áR(h\177&~?\037RÛ1/";
   int i;

   for(i = 0; cadena[i]; i++)
      if(isprint(cadena[i]))
         printf("%c, %d\n", cadena[i], iscntrl(cadena[i]));
      else
         printf("%d, %d\n", cadena[i], iscntrl(cadena[i]));

   return 0;
}
```

# Macro isdigit ANSI C

**Librería: ctype**

```
int isdigit(int c);
```

Comprueba sin un carácter es un dígito decimal.

isdigit es una macro que verifica el entero c pertenece al rango caracteres de digitos decimales, que depende de la categoría local LC_CTYPE, por defecto, el rango es de (*'0*' a *'9'*).

## Valor de retorno:

El valor de retorno será no nulo si c es un dígito decimal.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0ñfáR(4h&~?RÛ1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isdigit(cadena[i]));

   return 0;
}
```

# Macro isgraph ANSI C

**Librería: ctype**

```
int isgraph(int c);
```

Comprueba sin un carácter es imprimible.

isgraph es una macro que verifica el entero c pertenece al rango de caracteres con representación gráfica, que por defecto son todos menos el espacio ' '. El comportamiento depende de la categoría local de LC_CTYPE.

## Valor de retorno:

El valor de retorno será no nulo si c es un carácter gráfico.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0 ñsáR(h &~?RÛ 1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isgraph(cadena[i]));

   return 0;
}
```

# Macro islower ANSI C

**Librería: ctype**

```
int islower(int c);
```

Comprueba sin un carácter es de tipo minúscula.

islower es una macro que verifica el entero c pertenece al rango de caracteres de letras minúsculas, que por defecto son los que están en el rango *a* a *z*. El comportamiento depende de la categoría local de LC_CTYPE.

## Valor de retorno:

El valor de retorno será no nulo si c es un carácter en minúscula.

## Ejemplo:

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

int main()
{
   char cadena[] = "aAbBcCdD31&75$/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], islower(cadena[i]));

   return 0;
}
```

# Macro isprint ANSI C

**Librería: ctype**

```
int isprint(int c);
```

Comprueba sin un carácter es imprimible.

isgraph es una macro que verifica el entero c pertenece al rango de caracteres imprimibles, que por defecto son todos los caracteres imprimibles, incluido el espacio ' '. El comportamiento depende de la categoría local de LC_CTYPE.

## Valor de retorno:

El valor de retorno será no nulo si c es un carácter imprimible.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0 ñ\003sáR(h &~?\177RÛ 1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isprint(cadena[i]));

   return 0;
}
```

# Macro ispunct ANSI C

**Librería: ctype**

```
int ispunct(int c);
```

Comprueba sin un carácter es correspondiente a un signo de puntuación.

ispunct es una macro que verifica el entero c pertenece al rango de caracteres de los signos de puntuación, que por defecto son todos menos los alfanuméricos y el blanco ' '. El comportamiento depende de la categoría local de LC_CTYPE.

## Valor de retorno:

El valor de retorno será no nulo si c es un signo e puntuación.

## Ejemplo:

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

int main()
{
   char cadena[] = "aAb.Bc/Cd(D3:1&,75%$/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], ispunct(cadena[i]));

   return 0;
}
```

# Macro isspace ANSI C

**Librería: ctype**

```
int isspace(int c);
```

Comprueba sin un carácter es de tipo espacio.

isspace es una macro que verifica el entero c pertenece grupo de caracteres de espacio, ' ', tab, retorno de carro, nueva línea, tabulador vertical o salto de página. El comportamiento depende de la categoría local de LC_CTYPE.

## Valor de retorno:

El valor de retorno será no nulo si c es un carácter de tipo espacio.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0 ñsáR\n(h &~?\177R\tÛ 1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isspace(cadena[i]));

   return 0;
}
```

# Macro isupper ANSI C

**Librería: ctype**

```
int isupper(int c);
```

Comprueba sin un carácter es de tipo mayúscula.

islower es una macro que verifica el entero c pertenece al rango de caracteres de letras mayúsculas, que por defecto son los que están en el rango *A* a *Z*. El comportamiento depende de la categoría local de LC_CTYPE.

## Valor de retorno:

El valor de retorno será no nulo si c es un carácter en mayúscula.

## Ejemplo:

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

int main()
{
   char cadena[] = "aAbBcCdD31&75$/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isupper(cadena[i]));

   return 0;
}
```

# Macro isxdigit ANSI C

**Librería: ctype**

```
int isxdigit(int c);
```

Comprueba sin un carácter es un dígito hexadecimal.

isxdigit es una macro que verifica el entero c pertenece al rango caracteres de digitos decimales, que depende de la categoría local LC_CTYPE, por defecto, el rango es de (*'0'* a *'9'*, *'a'* a *'f'* y *'A'* a *'F'*).

## Valor de retorno:

El valor de retorno será no nulo si c es un dígito hexadecimal.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0ñfáR(4h&5d~?ERÛ1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %d\n", cadena[i], isxdigit(cadena[i]));

   return 0;
}
```

# Macros LC_ALL, LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME ANSI C

**Librería: locale**

```
#define LC_ALL
#define LC_COLLATE
#define LC_CTYPE
#define LC_MONETARY
#define LC_NUMERIC
#define LC_TIME
```

Estas expanden a expresiones constantes enteras con valores únicos, apropiados para uso como el primer argumento de la función setlocale.

## Ejemplo:

**DJGPP**

```
#define LC_ALL          0x1f
#define LC_COLLATE      0x01
#define LC_CTYPE        0x02
#define LC_MONETARY     0x04
#define LC_NUMERIC      0x08
#define LC_TIME         0x10
```

**Borland**

```
#define LC_ALL      0
#define LC_COLLATE  1
#define LC_CTYPE    2
#define LC_MONETARY 3
#define LC_NUMERIC  4
#define LC_TIME     5
```

```
#define LC_ALL          0
#define LC_COLLATE      1
#define LC_CTYPE        2
#define LC_MONETARY     3
#define LC_NUMERIC      4
#define LC_TIME         5
```

**Dev-C++**

# Macro L_tmpnam ANSI C

**Librería: stdio**

```
#define L_tmpnam
```

Una expresión constante entera que es el tamaño necesitado para el array de tipo **char** suficientemente grande para guardar una cadena del nombre de fichero temporal generada por la función tmpnam.

## Ejemplo:

```
#define L_tmpnam 260
```

**DJGPP**

```
#define L_tmpnam 13
```

**Borland**

```
#define L_tmpnam (260)
```

**Dev-C++**

# Macro MB_CUR_MAX ANSI C

**Librería: stdlib**

```
#define MB_CUR_MAX
```

Una expresión entera positiva cuyo valor es el número máximo de bytes en un carácter multibyte para el conjunto de caracteres extendidos especificado por la localización actual (categoría LC_CTYPE), y cuyo valor no es nunca mayor que MB_LEN_MAX.

## Ejemplo:

```
#define MB_CUR_MAX 1
```

**DJGPP**

```
#define MB_CUR_MAX 1
```

**Borland**

```
SIN DEFINIR
```

**Dev-C++**

# Macro NULL ANSI C

**Librería: locale**

```
#define NULL
```

Un puntero nulo constante definido según la implementación.

## Ejemplo:

```
#define NULL 0
```

**DJGPP**

```
#define NULL 0
ó
#define NULL 0L
ó
#define NULL ((void *)0)
```

**Borland**

```
#define NULL ((void*0)  /* Para C++ */
```

**Dev-C++**

# Macro NULL ANSI C

**Librería: stddef**

```
#define NULL
```

Un puntero nulo constante definido según la implementación.

## Ejemplo:

```
#define NULL 0
```

**DJGPP**

```
#define NULL 0
ó
#define NULL 0L
ó
#define NULL ((void *)0)
```

**Borland**

```
#define NULL ((void*0)  /* Para C++ */
```

**Dev-C++**

# Macro NULL ANSI C

**Librería: stdio**

```
#define NULL
```

Un puntero nulo constante definido según la implementación.

## Ejemplo:

| | |
|---|---|
| #define NULL 0 | **DJGPP** |
| #define NULL 0 | **Borland** |
| ó | |
| #define NULL 0L | |
| ó | |
| #define NULL ((void *)0) | |
| #define NULL ((void*0) /* Para C++ */ | **Dev-C++** |

# Macro NULL ANSI C

**Librería: stdlib**

```
#define NULL
```

Un puntero nulo constante definido según la implementación.

## Ejemplo:

```
#define NULL 0
```

**DJGPP**

```
#define NULL 0
ó
#define NULL 0L
ó
#define NULL ((void *)0)
```

**Borland**

```
#define NULL ((void*0)  /* Para C++ */
```

**Dev-C++**

# Constante NULL ANSI C

**Librería: string**

```
#define NULL
```

Un puntero nulo constante que está definido según la implementación.

# Macro NULL ANSI C

**Librería: time**

```
#define NULL
```

Un puntero nulo constante definido según la implementación.

## Ejemplo:

```
#define NULL 0
```

**DJGPP**

```
#define NULL 0
ó
#define NULL 0L
ó
#define NULL ((void *)0)
```

**Borland**

```
#define NULL ((void*0)  /* Para C++ */
```

**Dev-C++**

# Macro offsetof ANSI C

**Librería: stddef**

```
#define offsetof(tipo, designador_miembro)
```

Expande a una expresión constante entera de tipo size_t, el valor del cual es el desplazamiento en bytes al miembro de la estructura (denominado **designador_miembro**) desde el comienzo de su tipo de estructura (denominado **tipo**). El **designador_miembro** será tal que dado **static *tipo* t;** entonces la expresión **& (t.*designador_miembro*)** evalúa a una dirección constante de memoria. (Si el miembro especificado es un campo de bits, el comportamiento no está definido).

## Ejemplo:

```
#define offsetof(s_type, mbr) ((size_t) &((s_type *)0)->mbr)
```

**DJGPP**

```
#define offsetof( s_name, m_name )  (size_t)&(((s_name _FAR *)0)->m_name)
```

**Borland**

```
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)
```

**Dev-C++**

# Macro RAND_MAX ANSI C

**Librería: stdlib**

```
#define RAND_MAX
```

Un Expresión constante de intervalo, el valor del cual es el valor máximo retorando por la función rand. El valor de la macro *RAND_MAX* será de al menos 32767.

## Ejemplo:

```
#define RAND_MAX        2147483647
```

**DJGPP**

```
#define RAND_MAX 0x7FFFU
```

**Borland**

```
#define RAND_MAX        0x7FFF
```

**Dev-C++**

# Macro SEEK_CUR, SEEK_END, SEEK_SET ANSI C

**Librería: stdio**

```
#define SEEK_CUR
#define SEEK_END
#define SEEK_SET
```

Unas expresiones enteras y constantes con valores únicos, apropiados para el uso como el tercer argumento de la función fseek.

## Ejemplo:

```
#define SEEK_CUR 0
#define SEEK_END 1
#define SEEK_SET  2
```

**DJGPP**

```
#define SEEK_CUR 1
#define SEEK_END 2
#define SEEK_SET  0
```

**Borland**

```
#define SEEK_CUR (1)
#define SEEK_END (2)
#define SEEK_SET  (0)
```

**Dev-C++**

# Macros SIG_xxx and SIGxxxx ANSI C

**Librería: signal**

```
#define SIG_DFL
#define SIG_ERR
#define SIGABRT
#define SIGFPE
#define SIGILL
#define SIGSEGV
#define SIGTERM
```

Estos expanden a expresiones constantes con valores únicos que tienen tipo compatible con el segundo argumento a y el valor de retorno de la función {f:signal}, y cuyo valor se compara inigualmente a la localización de cualquier función declarable; y los siguientes, cada cual expande a una expresión entera positiva y constante que es número de señal para la condición especificada:

| | |
|---|---|
| **SIGABRT** | Terminación anormal, tal como es iniciada por la función abort |
| **SIGFPE** | Una Operación aritmética errónea, tal como es dividir entre cero o una operación resultando en un "desbordamiento" |
| **SIGILL** | Detección de un proceso inválido de una función, tal como es una instrucción ilegal |
| **SIGINT** | Recibo de señal de atención interactiva |
| **SIGSEGV** | Acceso inválido a almacenaje |
| **SIGTERM** | Una petición a terminar enviado a un programa |

Una implemntación no necesita generar ninguna de estas señales, excepto como un resultado de llamadas explícitas a la función raise.

## Ejemplo:

**DJGPP**

```
#define SIG_DFL ((void (*)(int))(0))
#define SIG_ERR ((void (*)(int))(1))
#define SIG_IGN ((void (*)(int))(-1))

#define SIGABRT 288
#define SIGFPE  289
#define SIGILL  290
#define SIGSEGV 291
#define SIGTERM 292
#define SIGINT  295
```

**Borland**

```
#ifdef __cplusplus
typedef void _USERENTRY (_EXPFUNC * _CatcherPTR)(int);
#else
typedef void _USERENTRY (_EXPFUNC * _CatcherPTR)();
#endif

#define SIG_DFL ((_CatcherPTR)0)   /* Default action */
#define SIG_IGN ((_CatcherPTR)1)   /* Ignore action  */
#define SIG_ERR ((_CatcherPTR)-1)  /* Error return   */

#define SIGABRT        22
#define SIGFPE          8          /* Floating point trap */
#define SIGILL          4          /* Illegal instruction */
#define SIGINT          2
#define SIGSEGV        11      /* Memory access violation */
#define SIGTERM        15
```

**Dev-C++**

```
typedef void (*_p_sig_fn_t)(int nSig);

#define SIG_DFL ((_p_sig_fn_t) 0)
#define SIG_IGN ((_p_sig_fn_t) 1)
#define SIG_ERR ((_p_sig_fn_t) -1)

#define SIGINT     2 /* Interactive attention  */
#define SIGILL     4 /* Illegal instruction */
#define SIGFPE     8 /* Floating point error */
#define SIGSEGV   11 /* Segmentation violation */
#define SIGTERM   15 /* Termination request */
#define SIGABRT   22 /* Abnormal termination (abort) */
```

# Macro stderr ANSI C

**Librería: stdio**

```
#define stderr
```

Expresión de tipo "puntero a FILE" que apunta a un objeto FILE asociado al stream estándar de error.

## Ejemplo:

```
extern FILE __dj_stderr;
#define stderr (&__dj_stderr)
```

**DJGPP**

```
extern FILE _RTLENTRY _EXPDATA
_streams[];
#define stderr (&_streams[2])
```

**Borland**

```
#define STDERR_FILENO 2
extern FILE (*__imp__iob)[];
#define _iob (*__imp__iob)
#define stderr
(&_iob[STDERR_FILENO])
```

**Dev-C++**

# Macro stdin ANSI C

**Librería: stdio**

```
#define stdin
```

Expresión de tipo "puntero a FILE" que apunta a un objeto FILE asociado al stream estándar de entrada.

## Ejemplo:

```
extern FILE __dj_stdin;
#define stdin (&__dj_stdin)
```

**DJGPP**

```
extern FILE _RTLENTRY _EXPDATA
_streams[];
#define stdin (&_streams[0])
```

**Borland**

```
#define STDERR_FILENO 0
extern FILE (*__imp__iob)[];
#define _iob (*__imp__iob)
#define stdin
(&_iob[STDIN_FILENO])
```

**Dev-C++**

# Macro stdout ANSI C

**Librería: stdio**

```
#define stdout
```

Expresión de tipo "puntero a FILE" que apunta a un objeto FILE asociado al stream estándar de salida.

## Ejemplo:

```
extern FILE __dj_stdout;
#define stdout (&__dj_stdout)
```

**DJGPP**

```
extern FILE _RTLENTRY _EXPDATA
_streams[];
#define stdout (&_streams[1])
```

**Borland**

```
#define STDOUT_FILENO 1
extern FILE (*__imp__iob)[];
#define _iob (*__imp__iob)
#define stdout
(&_iob[STDOUT_FILENO])
```

**Dev-C++**

# Macro TMP_MAX ANSI C

**Librería: stdio**

```
#define TMP_MAX
```

Una expresión entera y constante que es el número mínimo de nombres únicos de ficheros que serán generados por la función tmpnam. El valor de la macro *TMP_MAX* será al menos 25.

## Ejemplo:

| | |
|---|---|
| #define TMP_MAX 999999 | **DJGPP** |
| #define TMP_MAX 0xFFFF | **Borland** |
| #define TMP_MAX | **Dev-C++** |

# Macro toascii ANSI C

**Librería: ctype**

```
int toascii(int c);
```

Convierte caracteres a formato ASCII.

toascii es una macro convierte el entero c a ASCII eliminando todos los bits menos los siete de menor peso. Eso proporciona un valor dentro del rango entre 0 y 127.

## Valor de retorno:

El valor de retorno será el valor de c convertido.

## Ejemplo:

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

int main()
{
   char cadena[] = ";0ásRñ(hü&Rç1/";
   int i;

   for(i = 0; cadena[i]; i++)
      printf("%c, %c\n", cadena[i], toascii(cadena[i]));

   return 0;
}
```

# Macro va_arg ANSI C

**Librería: stdarg**

```
tipo va_arg(va_list pa, tipo);
```

Expande a una expresión que tiene el tipo y valor del siguiente argumento en la llamada. El parámetro **pa** será igual al de *va_list argumento* inicializado por va_start. Cada invocación de *va_arg* modifica **pa** tal que los valores de los argumentos sucesivos son retornados por turno. El parámetro ***tipo*** es un nombre de un tipo especificado, tal que el tipo del puntero a un objeto que tiene el tipo especificado, puede ser obtenido simplemente mediante añadiendo al final un ***** a ***tipo***. Si no hay ningún argumento siguiente, o si ***tipo*** no es compatible con el tipo del siguiente argumento (promocionado según las promociones de argumentos por defecto), el compartamiento no está definido.

## Valor de retorno:

La primera invocación de la macro *va_arg* después de aquélla de la macro va_start retorna el valor del argumento después de ésa, especificado por **paramN**. Invocaciones sucesivas retornan los valores de los argumentos restantes en sucesión.

## Ejemplo:

**DJGPP**

```
#define __dj_va_rounded_size(T)  \
        (((sizeof (T) + sizeof (int) - 1) / sizeof (int)) * sizeof (int))
```

```
#define va_arg(ap, T) \
    (ap = (va_list) ((char *) (ap) + __dj_va_rounded_size (T)), \
    *((T *) (void *) ((char *) (ap) - __dj_va_rounded_size (T))))
```

**Borland**

```
#define __size(x) ((sizeof(x)+sizeof(int)-1) & ~(sizeof(int)-1))
#define va_arg(ap, type) (*(type _FAR *)(((*(char _FAR *_FAR \
  *)&(ap))+=__size(type))-(__size(type))))
```

**Dev-C++**

```
#define __va_argsiz(t) \
    (((sizeof(t) + sizeof(int) - 1) / sizeof(int)) * sizeof(int))
```

```
#define va_arg(ap, t) \
    (((ap) = (ap) + __va_argsiz(t)),  \
    *((t*) (void*) ((ap) - __va_argsiz(t))))
```

# Macro va_end ANSI C

**Librería: stdarg**

```
void va_end(va_list pa);
```

Facilita un regreso normal desde la función cuya lista de argumentos de variables fue referida por la expansión de va_start que inicializó **va_list pa**. La macro *va_end* puede modificar **pa** para que ya no sea usado (sin una invocación interventiva a va_start). Si no hay una invocación correspondiente de la macro va_start, o si la macro *va_end* no es invocada antes del regreso, el comportamiento no está definido.

## Valor de retorno:

Esta macro no retorna ningún valor.

## Ejemplo:

```
#define va_end(ap)
```

**DJGPP**

```
#define va_end(ap) ((void)0)
```

**Borland**

```
#define va_end(ap) ((void)0)
```

**Dev-C++**

# Tipo va_list ANSI C

**Librería: stdarg**

```
typedef <tipo> va_list;
```

Un tipo apropiado para guardar información necesitada para las macros va_start, va_arg, y va_end. Si acceso a los varios argumentos es deseado, la función llamada declarará un objeto (denominado como **pa**, parámetro de argumentos, en esta referencia) teniendo tipo *va_list*. El objeto **pa** puede ser pasado como un argumento a otra función; si esa función invoca la macro va_arg con el parámetro **pa**, el valor de **pa** en la función llamada es determinado y será pasado a la macro va_end anterior a cualquier referencia posterior a **pa**.

## Ejemplo:

```
#define __DJ_va_list typedef void *va_list;
```

**DJGPP**

```
typedef void _FAR *va_list;
```

**Borland**

```
typedef char* va_list;
```

**Dev-C++**

# Macro va_start ANSI C

**Librería: stdarg**

```
void va_start(va_list pa, paramN);
```

Será invocada antes de cualquier acceso a los argumentos sin nombre. La macro *va_start* inicializa **pa** para uso posterior en va_arg y va_end. El parámetro ***paramN*** es el identificador del parámetro más a la derecha en la lista de parámetros de variables en la definición de la función (justo antes de **,...**). Si el parámetro ***paramN*** es declarado con la clase de almacenamiento **register**, con un tipo de función o array, o con un tipo que no es compatible con el tipo que resulta después de la aplicación de las promociones de argumentos por defecto, el comportamiento no está definido.

## Valor de retorno:

La macro *va_start* no retorna ningún valor.

## Ejemplo:

**DJGPP**

```
#define __dj_va_rounded_size(T)  \
      (((sizeof (T) + sizeof (int) - 1) / sizeof (int)) * sizeof (int))
```

```
#define va_start(ap, last_arg) \
      (ap = ((va_list) __builtin_next_arg (last_arg)))
```

**Borland**

```
#define __size(x) ((sizeof(x)+sizeof(int)-1) & ~(sizeof(int)-1))
```

```
#define va_start(ap, parmN) ((void)((ap) = \
      (va_list)((char _FAR *)(&parmN)+__size(parmN))))
```

**Dev-C++**

```
#define __va_argsiz(t) \
      (((sizeof(t) + sizeof(int) - 1) / sizeof(int)) * sizeof(int))
```

```
#define va_start(ap, pN) \
      ((ap) = ((va_list) (&pN) + __va_argsiz(pN)))
```

# Macro _IOFBF, _IOLBF, _IONBF ANSI C

**Librería: stdio**

```
#define _IOFBF
#define _IOLBF
#define _IONBF
```

Son expresiones enteras constantes con valores únicos, apropiados para el uso como el tercer argumento para la función setvbuf.

## Ejemplo:

| | |
|---|---|
| #define _IOFBF 00001<br>#define _IOLBF 00004<br>#define _IONBF 00002 | **DJGPP** |
| #define _IOFBF 0<br>#define _IOLBF 1<br>#define _IONBF 2 | **BORLAND** |
| #define _IOFBF 0<br>#define _IOLBF 1<br>#define _IONBF 2 | **Dev-C++** |

# Indice de tipos y estructuras

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## -C-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| clock_t | time | time.h | ctime |

## -D-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| div_t | stdlib | stdlib.h | cstdlib |

## -F-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| FILE | stdio | stdio.h | cstdio |
| fpos_t | stdio | stdio.h | cstdio |

## -J-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| jmp_buf | setjmp | setjmp.h | csetjmp |

## -L-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| lconv | locale | locale.h | clocale |
| ldiv_t | stdlib | stdlib.h | cstdlib |

## -P-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| ptrdiff_t | stddef | stddef.h | cstddef |

## -S-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| sig_atomic_t | signal | signal.h | csignal |
| size_t | stddef | stddef.h | cstddef |
| size_t | stdio | stdio.h | cstdio |
| size_t | stdlib | stdlib.h | cstdlib |
| size_t | string | string.h | cstring |
| size_t | time | time.h | ctime |

## -T-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| time_t | time | time.h | ctime |
| tm | time | time.h | ctime |

## -W-

| Estructura | Librería | Fichero de cabecera C | Fichero de cabecera C++ |
|---|---|---|---|
| wchar_t | stddef | stddef.h | cstddef |
| wchar_t | stdlib | stdlib.h | cstdlib |

# Tipo clock_t ANSI C

**Librería: time**

```
typedef <tipo> clock_t;
```

Un tipo aritmético capaz de representar el tiempo.

## Ejemplo:

```
typedef int clock_t;
```

**DJGPP**

```
typedef long clock_t;
```

**Borland**

```
typedef long clock_t;
```

**Dev-C++**

# Tipo div_t ANSI C

**Librería: stdlib**

```
typedef <tipo> div_t;
```

Un tipo de estructura que es el tipo del valor retornado por la función div.

## Ejemplo:

**DJGPP**

```
typedef struct {
   int quot;
   int rem;
} div_t;
```

**Borland**

```
typedef struct {
   int   quot;
   int   rem;
} div_t;
```

**Dev-C++**

```
typedef struct { int quot, rem; } div_t;
```

# Tipo FILE ANSI C

**Librería: stdio**

```
typedef <tipo> FILE;
```

Un tipo de objeto capaz de almacenar toda la información necesitada para controlar un stream, incluyendo su indicador de posición de fichero, un puntero a su almacenamiento asociado (si existe), un indicador de errores que registra si se ha producido un error de lectura y/o ecsritura, y un indicador de final de fichero que registra si se ha llegado al final del fichero.

## Ejemplo:

**DJGPP**

```
typedef struct {
   int     _cnt;
   char *  _ptr;
   char *  _base;
   int     _bufsiz;
   int     _flag;
   int     _file;
   char *  _name_to_remove;
   int     _fillsize;
} FILE;
```

**Borland**

```
typedef struct {
   int           level;
   unsigned      flags;
   char          fd;
   unsigned char hold;
   int           bsize;
   unsigned char _FAR *buffer;
   unsigned char _FAR *curp;
   unsigned      istemp;
   short         token;
} FILE;
```

**Dev-C++**

```
typedef struct _iobuf
{
   char*   _ptr;
   int     _cnt;
   char*   _base;
   int     _flag;
   int     _file;
   int     _charbuf;
   int     _bufsiz;
   char*   _tmpfname;
} FILE;
```

# Tipo fpos_t ANSI C

**Librería: stdio**

```
typedef <tipo> fpos_t;
```

Un tipo de objeto capaz de almacenar toda la información necesitada para especificar únicamente cada posición dentro del fichero.

## Ejemplo:

| | |
|---|---|
| typedef unsigned long fpos_t; | **DJGPP** |
| typedef long fpos_t; | **Borland** |
| typedef long fpos_t; | **Dev-C++** |

# Tipo jmp_buf ANSI C

**Librería: setjmp**

```
typedef <tipo> jmp_buf;
```

Un tipo de array/arreglo apropiado para guardar la información necesaria para restaurar un entorno de llamada.

## Ejemplo:

**DJGPP**

```
typedef struct __jmp_buf {
   unsigned long __eax, __ebx, __ecx, __edx, __esi;
   unsigned long __edi, __ebp, __esp, __eip, __eflags;
   unsigned short __cs, __ds, __es, __fs, __gs, __ss;
   unsigned long __sigmask; /* for POSIX signals only */
   unsigned long __signum; /* for expansion */
   unsigned long __exception_ptr; /* pointer to previous exception */
   unsigned char __fpu_state[108]; /* for future use */
} jmp_buf[1];
```

**Borland**

```
typedef struct __jmp_buf {
   unsigned    j_sp;
   unsigned    j_ss;
   unsigned    j_flag;
   unsigned    j_cs;
   unsigned    j_ip;
   unsigned    j_bp;
   unsigned    j_di;
   unsigned    j_es;
   unsigned    j_si;
   unsigned    j_ds;
   #if !defined(__TINY__)
   unsigned    j_excep;
   unsigned    j_context;
   #endif /* !__TINY__ */
} jmp_buf[1];
```

**Dev-C++**

```
#define _JBLEN 16
#define _JBTYPE int
typedef _JBTYPE jmp_buf[_JBLEN];
```

# Estructura global lconv ANSI C

**Librería: locale**

```
struct lconv {...};
```

Contiene miembros relacionados con el formateo de valores numéricos. La estructura contendrá al menos los siguientes miembros, en cualquier orden. En la localidad "**C**", los miembros tendrán los valores especificados en los comentarios.

```
char *decimal_point;       /* "." */
char *thousands_sep;       /* "" */
char *grouping;            /* "" */
char *int_curr_symbol;     /* "" */
char *currency_symbol;     /* "" */
char *mon_decimal_point;   /* "" */
char *mon_thousands_sep;   /* "" */
char *mon_grouping;        /* "" */
char *positive_sign;       /* "" */
char *negative_sign;       /* "" */
char int_frac_digits;      /* CHAR_MAX */
char frac_digits;          /* CHAR_MAX */
char p_cs_precedes;        /* CHAR_MAX */
char p_sep_by_space;       /* CHAR_MAX */
char n_cs_precedes;        /* CHAR_MAX */
char n_sep_by_space;       /* CHAR_MAX */
char p_sign_posn;          /* CHAR_MAX */
char n_sign_posn;          /* CHAR_MAX */
```

## Ejemplo:

**DJGPP**

```
struct lconv {
  char *currency_symbol;
  char *decimal_point;
  char *grouping;
  char *int_curr_symbol;
  char *mon_decimal_point;
  char *mon_grouping;
  char *mon_thousands_sep;
  char *negative_sign;
  char *positive_sign;
  char *thousands_sep;
  char frac_digits;
  char int_frac_digits;
  char n_cs_precedes;
  char n_sep_by_space;
  char n_sign_posn;
  char p_cs_precedes;
  char p_sep_by_space;
  char p_sign_posn;
};
```

**Borland**

```
struct lconv {
   char *decimal_point;
   char *thousands_sep;
   char *grouping;
   char *int_curr_symbol;
   char *currency_symbol;
   char *mon_decimal_point;
   char *mon_thousands_sep;
   char *mon_grouping;
   char *positive_sign;
   char *negative_sign;
   char int_frac_digits;
   char frac_digits;
   char p_cs_precedes;
   char p_sep_by_space;
   char n_cs_precedes;
   char n_sep_by_space;
   char p_sign_posn;
   char n_sign_posn;
};
```

```
struct lconv
{
        char*   decimal_point;
        char*   thousands_sep;
        char*   grouping;
        char*   int_curr_symbol;
        char*   currency_symbol;
        char*   mon_decimal_point;
        char*   mon_thousands_sep;
        char*   mon_grouping;
        char*   positive_sign;
        char*   negative_sign;
        char    int_frac_digits;
        char    frac_digits;
        char    p_cs_precedes;
        char    p_sep_by_space;
        char    n_cs_precedes;
        char    n_sep_by_space;
        char    p_sign_posn;
        char    n_sign_posn;
};
```

**Dev-C++**

# Tipo ldiv_t ANSI C

**Librería: stdlib**

```
typedef <tipo> ldiv_t;
```

Un tipo de estructura que es el tipo del valor retornado por la función ldiv.

## Ejemplo:

**DJGPP**

```
typedef struct {
   long quot;
   long rem;
} ldiv_t;
```

**Borland**

```
typedef struct {
   long   quot;
   long   rem;
} ldiv_t;
```

**Dev-C++**

```
typedef struct { long quot, rem; } ldiv_t;
```

# Tipo ptrdiff_t ANSI C

**Librería: stddef**

```
typedef <tipo> ptrdiff_t;
```

Un tipo entero con signo del resultado de restar dos punteros.

## Ejemplo:

**DJGPP**

```
typedef int ptrdiff_t;
```

**Borland**

```
#if     defined(__LARGE__) || defined(__HUGE__) || defined(__COMPACT__)
typedef long    ptrdiff_t;
#else
typedef int     ptrdiff_t;
#endif
```

**Dev-C++**

```
#define __PTRDIFF_TYPE__ long int
typedef __PTRDIFF_TYPE__ ptrdiff_t;
```

# Tipo sig_atomic_t ANSI C

**Librería: signal**

```
typedef <tipo> sig_atomic_t;
```

El tipo entero de un objeto que puede ser accedido como una entidad atómica, incluso en la presencia de interruptores asíncronos.

## Ejemplo:

```
typedef int sig_atomic_t;
```

**DJGPP**

```
typedef int sig_atomic_t;
```

**Borland**

```
typedef int sig_atomic_t;
```

**Dev-C++**

# Tipo size_t ANSI C

**Librería: stddef**

```
typedef <tipo> size_t;
```

El tipo entero sin signo que es el resultado del operador **sizeof**.

## Ejemplo:

```
typedef long unsigned int size_t;
```
**DJGPP**

```
typedef unsigned size_t;
```
**Borland**

```
typedef long unsigned int size_t;
```
**Dev-C++**

# Tipo size_t ANSI C

**Librería: stdio**

```
typedef <tipo> size_t;
```

El tipo entero sin signo que es el resultado del operador **sizeof**.

## Ejemplo:

| | |
|---|---|
| typedef long unsigned int size_t; | **DJGPP** |
| typedef unsigned size_t; | **Borland** |
| typedef long unsigned int size_t; | **Dev-C++** |

# Tipo size_t no encontrada.

Lo sentimos mucho, pero la función consultada aún no ha sido incluída en la documentación.

# Tipo size_t ANSI C

**Librería: string**

```
typedef unsigned size_t;
```

Tipo de un número entero sin signo.

# Tipo size_t ANSI C

**Librería: time**

```
typedef <tipo> size_t;
```

El tipo entero sin signo del resultado del operador **sizeof**.

## Ejemplo:

```
typedef long unsigned int size_t;
```

**DJGPP**

```
typedef unsigned size_t;
```

**Borland**

```
typedef long unsigned int size_t;
```

**Dev-C++**

# Tipo time_t ANSI C

**Librería: time**

```
typedef <tipo> time_t;
```

Un tipo aritmético capaz de representar el tiempo.

## Ejemplo:

```
typedef unsigned int time_t;
```

**DJGPP**

```
typedef long time_t;
```

**Borland**

```
typedef long time_t;
```

**Dev-C++**

# Struct tm ANSI C

**Librería: time**

```
struct tm {...};
```

Guarda los componentes de la hora y la fecha, en formato separado. La estructura contendrá al menos los siguientes miembros, en cualquier orden. La semántica de los miembros y sus intervalos normales están expresados en los comentarios.

```
int tm_sec;         /* los segundos después del minuto -- [0,61] */
int tm_min;         /* los minutos después de la hora -- [0,59] */
int tm_hour;        /* las horas desde la medianoche -- [0,23] */
int tm_mday;        /* el día del mes -- [1,31] */
int tm_mon;         /* los meses desde Enero -- [0,11] */
int tm_year;        /* los años desde 1900 */
int tm_wday;        /* los días desde el Domingo -- [0,6] */
int tm_yday;        /* los días desde Enero -- [0,365] */
int tm_isdst;       /* el flag del Horario de Ahorro de Energía */
```

El valor de **tm_isdst** es positivo si el Horario de Ahorro de Energía está en efecto, cero si no lo está, y negativo si la información no está disponible.

## Ejemplo:

**DJGPP**

```
struct tm {
  int tm_sec;
  int tm_min;
  int tm_hour;
  int tm_mday;
  int tm_mon;
  int tm_year;
  int tm_wday;
  int tm_yday;
  int tm_isdst;
  char *__tm_zone;
  int __tm_gmtoff;
};
```

**Borland**

```
struct tm
{
  int   tm_sec;
  int   tm_min;
  int   tm_hour;
  int   tm_mday;
  int   tm_mon;
  int   tm_year;
  int   tm_wday;
  int   tm_yday;
  int   tm_isdst;
};
```

**Dev-C++**

```
struct tm
{
        int     tm_sec;         /* Seconds: 0-59 (K&R says 0-61?) */
        int     tm_min;         /* Minutes: 0-59 */
        int     tm_hour;        /* Hours since midnight: 0-23 */
        int     tm_mday;        /* Day of the month: 1-31 */
        int     tm_mon;         /* Months *since* january: 0-11 */
        int     tm_year;        /* Years since 1900 */
        int     tm_wday;        /* Days since Sunday (0-6) */
        int     tm_yday;        /* Days since Jan. 1: 0-365 */
        int     tm_isdst;       /* +1 Daylight Savings Time, 0 No DST,
                                 * -1 don't know */
};
```

# Tipo wchar_t ANSI C

**Librería: stddef**

```
typedef <tipo> wchar_t;
```

Un tipo entero cuyo intervalo de valores puede representar códigos únicos para todos los miembros del conjunto más grande de caracteres extendidos especificado de entre las localidades soportadas; el carácter nulo tendrá el número de código, cero, y cada miembro del conjunto básico de caracteres el número de código igual a su valor cuando usado como el carácter único en una constante de caracteres enteros.

## Ejemplo:

**DJGPP**

```
/* Para C */
#define __DJ_wchar_t    typedef unsigned short wchar_t;
/* Para C++, ya que es un tipo "verdadero" */
#define __DJ_wchar_t
```

**Borland**

```
typedef unsigned short wchar_t;
```

**Dev-C++**

```
#define __WCHAR_TYPE__ int
#endif
#ifndef __cplusplus
typedef __WCHAR_TYPE__ wchar_t;
#endif
```

# Tipo wchar_t ANSI C

**Librería: stdlib**

```
typedef <tipo> wchar_t;
```

Un tipo entero cuyo intervalo de valores puede representar códigos únicos para todos los miembros del conjunto más grande de caracteres extendidos especificado de entre las localidades soportadas; el carácter nulo tendrá el número de código, cero, y cada miembro del conjunto básico de caracteres el número de código igual a su valor cuando usado como el carácter único en una constante de caracteres enteros.

## Ejemplo:

**DJGPP**

```
/* Para C */
#define __DJ_wchar_t    typedef unsigned short wchar_t;
/* Para C++, ya que es un tipo "verdadero" */
#define __DJ_wchar_t
```

**Borland**

```
typedef unsigned short wchar_t;
```

**Dev-C++**

```
#define __WCHAR_TYPE__ int
#endif
#ifndef __cplusplus
typedef __WCHAR_TYPE__ wchar_t;
#endif
```