Funciones de “C” estándar clasificadas por ficheros de cabecera.

<ctype.h>

Incluye algunas funciones útiles para la clasificación y el mapeado de códigos.
Cada función acepta un argumento de tipo "int" cuyo valor puede ser el valor de la macro "EOF" o cualquier valor representable por el tipo "unsigned char". Esto es, el argumento puede ser el valor devuelto por "fgetc", "fputc", "getc", "getchar", "putc", "putchar", "tolower", "toupper" o "ungetc" (declaradas en "<stdio.h>").

isalnum
int isalnum (int c);
Devuelve un valor distinto de cero si "c" es una letra minúscula "a-z" o mayúscula "A-Z", uno de los dígitos decimales "0-9" o cualquier otro carácter alfabético local.

isalpha
int isalpha (int c);
Devuelve un valor distinto de cero si "c" es una letra minúscula "a-z" o mayúscula "A-Z", o cualquier otro carácter alfabético local.

iscntrl
int iscntrl (int c);
Devuelve un valor distinto de cero si "c" es cualquier carácter de control (como FF, HT, NL).

isdigit
int isdigit (int c);
Devuelve un valor distinto de cero si "c" es cualquiera de los dígitos decimales (0-9).

isgraph
int isgraph (int c);
Devuelve un valor distinto de cero si "c" es cualquier carácter de impresión excepto "espacio".

islower
int islower (int c);
Devuelve un valor distinto de cero si "c" es cualquiera de las letras minúsculas "a-z" u otra minúscula local.

isprint
int isprint (int c);
Devuelve un valor distinto de cero si "c" es cualquier carácter imprimible, incluyendo el "espacio".

ispunct
int ispunct (int c);
Devuelve un valor distinto de cero si "c" es cualquier carácter imprimible excepto "espacio", o si "isalnum(c)" es distinto de cero.

isspace
int isspace (int c);
Devuelve un valor distinto de cero si "c" es "CR", "FF", "HT", "NL", "VT", "espacio" o cualquier otro carácter de separación local.

isupper
int isupper (int c);
Devuelve un valor distinto de cero si "c" es una de las letras mayúsculas "A-Z" u otra mayúscula local.

isxdigit
isxdigit (int c);
Devuelve un valor distinto de cero si "c" es cualquier dígito hexadecimal "0-9", "A-F", "a-f".

tolower
tolower (int c);
Devuelve la correspondiente letra minúscula si existe y si "isupper(c)" es distinto de cero; en caso contrario, devuelve "c".

toupper
int toupper (int c);
Devuelve la correspondiente letra mayúscula si existe y si "islower(c)" es distinto de cero; en caso contrario, devuelve "c".

<errno.h>

Permite comprobar el valor almacenado en "errno" por algunas funciones de librería.
Al arrancar el programa, el valor almacenado en "errno" es cero. Las funciones de librería almacenan sólo valores mayores que 0 en "errno".
Para comprobar si una función almacena un valor en "errno", el programa debería almacenar un 0 en "errno" antes de llamar a la función.

EDOM
#define EDOM <expresion #if>
Almacena un valor en "errno" según exista o no un error de dominio.

ERANGE
#define ERANGE <expresion #if>
Almacena un valor en "errno" según exista o no un error de rango.

errno
#define errno <valor modificable int>
Designa un objeto de datos al que se asigna un valor mayor que cero dependiendo de ciertos errores.

<float.h>

Establece algunas propiedades de las representaciones de tipo real.

DBL_DIG
#define DBL_DIG <valor_entero 10>
Número de dígitos de precisión para el tipo "double".

DBL_EPSILON
#define DBL_EPSILON <valor_double 10^{-9}>
Produce el menor valor "x" de tipo "double", tal que 1.0 + x ? 1.0.

DBL_MANT_DIG
#define DBL_MANT_DIG <valor_int>
Produce el número de dígitos de mantisa, base "FLT_RADIX", para el tipo "double".

DBL_MAX
#define DBL_MAX <valor_double 10^{37}>
Valor representable finito más grande de tipo "double".

DBL_MAX_10_EXP
#define DBL_MAX_10_EXP <valor_int 37>
Máximo entero "x" tal que 10 elevado a "x" sea un valor representable finito de tipo "double".

DBL_MAX_EXP
#define DBL_MAX_EXP <valor_int>
Máximo entero "x" tal que "FLT_RADIX" elevado a "x-1" es un valor finito representable de tipo "double".

DBL_MIN
#define DBL_MIN <valor_double 10^{-37}>
Valor finito más pequeño representable normalizado de tipo "double".

DBL_MIN_10_EXP
#define DBL_MIN_10_EXP <valor_int -37>
Mínimo entero "x" tal que 10 elevado a "x" es un valor finito representable y normalizado de tipo "double".

DBL_MIN_EXP
#define DBL_MIN_EXP <valor_int>
Mínimo entero "x" tal que "FLT_RADIX" elevado a "x-1" es un valor finito representable y normalizado de tipo "double".

FLT_DIG
#define FLT_DIG <valor_int 6>
Número de dígitos decimales de precisión para el tipo "float".

FLT_EPSILON
#define FLT_EPSILON <valor_float 10^{-5}>
Valor más pequeño "x" de tipo "float", tal que 1.0 + x ? 1.0.

FLT_MANT_DIG
#define FLT_MANT_DIG <valor_int>
Número de dígitos de mantisa, en base "FLT_RADIX", para el tipo "float".

FLT_MAX
#define FLT_MAX <valor_float 10^{37}>
Valor finito representable más grande de tipo "float".

FLT_MAX_10_EXP
#define FLT_MAX_10_EXP <valor_int 37>
Máximo entero "x" tal que 10 elevado a "x" es un valor finito representable de tipo "float".

FLT_MAX_EXP
#define FLT_MAX_EXP <valor_int>
Máximo entero "x" tal que "FLT_RADIX" elevado a "x-1" es un valor finito representable de tipo "float".

FLT_MIN
#define FLT_MIN <valor_float 10^{-37}>
Menor valor finito representable y normalizado de tipo "float".

FLT_MIN_10_EXP
#define FLT_MIN_10_EXP <valor_int -37>
Mínimo entero "x" tal que 10 elevado a "x" es un valor finito representable y normalizado de tipo "float".

FLT_MIN_EXP
#define FLT_MIN_EXP <valor_int>
Mínimo ntero "x" tal que "FLT_RADIX" elevado a "x-1" es un valor finito representable y normalizado de tipo "float".

FLT_RADIX
#define FLT_RADIX <expresion #if 2>
Produce la base de numeración de todas las representaciones reales.

FLT_ROUNDS
#define FLT_ROUNDS <valor_int>
Describe el modo de redondeo para las operaciones con reales. El valor es "-1" si el modo está indeterminado, es "0" si el redondeo es hacia "0", es "1" si el redondeo es hacia el valor representable más cercano, es "2" si el redondeo es hacia +8 y es "3" si el redondeo es hacia -8 .

LDBL_DIG
#define LDBL_DIG <valor_int 10>
Número de dígitos decimales de precisión para el tipo "long double".

LDBL_EPSILON
#define LDBL_EPSILON <valor_long_double 10^{-9}>
Menor valor de "x" de tipo "long double" tal que 1.0 + x ? 1.0.

LDBL_MANT_DIG
#define LDBL_MANT_DIG <valor_int>
Número de dígitos de mantisa, en base "FLT_RADIX", para el tipo "long double".

LDBL_MAX
#define LDBL_MAX <valor_long_double 10^{37}>
Valor finito representable más grande de tipo "long double".

LDBL_MAX_10_EXP
#define LDBL_MAX_10_EXP <valor_int 37>
Máximo entero "x" tal que 10 elevado a "x" es un valor finito representable de tipo "long double".

LDBL_MAX_EXP
#define LDBL_MAX_EXP <valor_int>
Máximo entero "x" tal que "FLT_RADIX" elevado a "x-1" es un valor finito representable de tipo "long double".

LDBL_MIN
#define LDBL_MIN <valor_long_double 10^{-37}>
Menor valor finito representable y normalizado de tipo "long double".

LDBL_MIN_10_EXP
#define LDBL_MIN_10_EXP <valor_int -37>
Mínimo entero "x" tal que 10 elevado a "x" es unvalor finito representable y normalizado de tipo "long double".

LDBL_MIN_EXP
#define LDBL_MIN_EXP <valor_int>
Mínimo entero "x" tal que "FLT_RADIX" elevado a "x-1" es un valor finito representable y normalizado de tipo "long double".

<limits.h>

Contiene macros que determinan varias propiedades de las representaciones de tipos enteros.

CHAR_BIT
#define CHAR_BIT <expresion #if 8>
Número de bits usados en la representación de un objeto de datos de tipo "char".

CHAR_MAX
#define CHAR_MAX <expresion #if 127>
Máximo valor para el tipo "char", que es el mismo que "SCHAR_MAX" si "char" representa valores negativos; en caso contrario, el valor es el mismo que "UCHAR_MAX".

CHAR_MIN
#define CHAR_MIN <expresion #if 0>
Mínimo valor para el tipo "char", el mismo que "SCHAR_MIN" si "char" representa valores negativos; en caso contrario, el valor es 0.

INT_MAX
#define INT_MAX <expresion #if 32767>
Máximo valor para el tipo "int".

INT_MIN
#define INT_MIN <expresion #if -32767>
Mínimo valor para el tipo "int".

LONG_MAX
#define LONG_MAX <expresion #if 2147483647>
Máximo valor para el tipo "long".

LONG_MIN
#define LONG_MIN <expresion #if -2147483647>
Mínimo valor para el tipo "long".

MB_LEN_MAX
#define MB_LEN_MAX <expresion #if 1>
Máximo número de caracteres que constituyen un carácter multibyte en un conjunto de caracteres local.

SCHAR_MAX
#define SCHAR_MAX <expresion #if 127>
Máximo valor para el tipo "signed char".

SCHAR_MIN
#define SCHAR_MIN <expresion #if -127>
Mínimo valor para el tipo "signed char".

SHRT_MAX
#define SHRT_MAX <expresion #if 32767>
Máximo valor para el tipo "short".

SHRT_MIN
#define SHRT_MIN <expresion #if -32767>
Mínimo valor para el tipo "short".

UCHAR_MAX
#define UCHAR_MAX <expresion #if 255>
Máximo valor para el tipo "unsigned char".

UINT_MAX
#define UINT_MAX <expresion #if 65535>
Máximo valor para el tipo "unsigned int".

ULONG_MAX
#define ULONG_MAX <expresion #if 4294967295>
Máximo valor para el tipo "unsigned long".

USHRT_MAX
#define USHRT_MAX <expresion #if 65535>
Máximo valor para el tipo "unsigned short".

<math.h>

Contiene la declaración de algunas funciones para realizar operaciones matemáticas comunes sobre valores de tipo "double".

Una excepción por error de dominio ocurre cuando la función no está definida para el valor o valores de sus argumentos. La función informa almacenando el valor "EDOM" en "errno" y devolviendo un valor peculiar propio de cada implementación.

Una excepción por error de rango ocurre cuando el valor de la función está definido pero no puede ser representado por un valor de tipo "double". La función informa almacenando el valor "ERANGE" en "errno" y devolviendo uno de los siguientes valores: "HUGE_VAL" si el valor es positivo y demasiado grande, "0" si el valor es demasiado pequeño para ser representado, "-HUGE_VAL" si el valor es negativo y con valor absoluto demasiado grande.

HUGE_VAL
#define HUGE_VAL <valor_double>
Valor devuelto por algunas funciones debido a un error de rango, que puede ser una representación de infinitud.

acos
double acos (double x);
Angulo cuyo coseno es "x", en el rango [0,p] radianes.

asin
double asin (double x);
Angulo cuyo seno es "x", en el rango [-p/2, +p/2] radianes.

atan
double atan (double x);
Angulo cuya tangente es "x", en el rango [-p/2, +p/2] radianes.

atan2
double atan2 (double y, double x);
Angulo cuya tangente es "y/x", en el rango [-p, +p] radianes.

ceil
double ceil (double x);
Valor entero más pequeño no menor que "x".

cos
double cos (double x);
Coseno de "x" (en radianes).

cosh
double cosh (double x);
Coseno hiperbólico de "x".

exp
double exp (double x);
Exponencial de "x", e^x.

fabs
double fabs (double x);
Valor absoluto de "x", "|x|".

floor
double floor (double x);
Mayor valor entero menor que "x".

fmod
double fmod (double x, double y);
Resto de "x/y", el cual es "x-i*y" para algún entero "i", tal que "i*y<x<(i+1)*y". Si "y" es cero, la función o informa de un error de dominio o simplemente devuelve el valor 0.

frexp
double frexp (double x, int *pexp);
Determina una fracción "f" y un entero "i" en base 2 que representa el valor de "x". Devuelve el valor "f" y almacena el entero "i" en "*pexp" tal que "f" está en el intervalo [1/2, 1) o tiene el valor 0, y "x" se iguala a "f*(2^i)". Si "x" es 0, "*pexp" es también 0.

ldexp
double ldexp (double x, int exponente);
Devuelve "$x*2^{exponente}$".

log
double log (double x);
Devuelve el logaritmo natural de "x".

log10
double log10 (double x);
Devuelve el logaritmo en base 10 de "x".

modf
double modf (double x, double *pint);
Determina un entero "i" más una fracción "f" que representan el valor de "x". Devuelve el valor "f" y almacena el entero "i" en "*pint", tal que "f+i" se iguala a "x", "|f|" está en el intervalo [0,1), y tanto "f" como "i" tienen el mismo signo que "x".

pow
double pow (double x, double y);
Devuelve "x" elevado a la potencia "y".

sin
double sin (double x);
Devuelve el seno de "x" (en radianes).

sinh
double sinh (double x);
Devuelve el seno hiperbólico de "x".

sqrt
double sqrt (double x);
Devuelve la raíz cuadrada de "x".

tan
double tan (double x);
Devuelve la tangente de "x" (en radianes).

tanh
double tanh (double x);
Devuelve la tangente hiperbólica de "x".

<stdarg.h>

Contiene declaraciones que permiten acceder a los argumentos adicionales sin nombre en una función que acepta un número variable de argumentos.

Para acceder a los argumentos adicionales, el programa debe ejecutar primero el macro "va_start" dentro del cuerpo de la función para inicializar un objeto de datos con información de contexto. Ejecutando sucesivamente el macro "va_arg", indicando esa información de contexto, se pueden ir obteniendo los valores de los argumentos adicionales en orden, empezando por el primer argumento sin nombre.
Se puede ejecutar "va_arg" desde cualquier función que pueda acceder a la información de contexto grabada por "va_start".
Si se ha ejecutado "va_start" en una función, se debe ejecutar el macro "va_end" en la misma función, indicando la misma información de contexto, antes del retorno de la función.
Para almacenar la información de contexto, se declara un objeto de datos "va_list", que puede ser un tipo "array" que afecta a cómo comparte información el programa con las funciones.

va_arg
#define va_arg (va_list ap, T) <valor_de_tipo_T>
Produce el valor del siguiente argumento en orden, según la información de contexto indicada por "ap". El argumento adicional debe ser de tipo "T" depués de aplicar las reglas para la promoción de argumentos en la ausencia de prototipo de función.

va_end
#define va_end (va_list ap) <expresion_void>
Realiza la limpieza necesaria para que la función pueda retornar.

va_list
typedef do-tipo va_list;
Es el tipo de datos "do_tipo" que se declara para contener la información de contexto inicializada por "va_start" y utilizada por "va_arg".

va_start
#define va_start (va_list ap, ultimo_arg) <expresion_void>
Almacena la información de contexto inicial en "ap". "ultimo_arg" es el nombre del último argumento declarado.

<stdio.h>

Incluye macros y funciones para realizar operaciones de entrada y salida sobre ficheros y flujos de datos.

_IOFBF
#define _IOFBF <expresion constante entera>
Valor del argumento "modo" para "setvbuf" para indicar "buffer" completo.

_IOLBF
#define _IOLBF <expresion constante entera>
Valor del argumento "modo" para "setvbuf" para indicar el "buffer" de línea.

_IONBF
#define _IONBF <expresion constante entera>
Valor del argumento "modo" para "setvbuf" para indicar que no se usa "buffer".

BUFSIZ
#define BUFSIZ <expresion constante entera 256>
Tamaño del "buffer" de flujo de datos utilizado por "setbuf".

EOF
#define EOF <expresion constante entera 0>
Valor de retorno utilizado para señalar el fin de fichero.

FILE
typedef d-tipo FILE;
Es el tipo de datos que almacena toda la información de control para un flujo de datos.

FILENAME_MAX
#define FILENAME_MAX <expresion constante entera>
Máximo tamaño de un "array" de caracteres para almacenar el nombre de un fichero.

FOPEN_MAX
#define FOPEN_MAX <expresion constante entera 8>
Máximo número de ficheros que el entorno operativo permite que estén abiertos simultáneamente (incluyendo "stderr", "stdin", "stdout").

L_tmpnam
#define L_tmpnam <expresion constante entera>
Número de caracteres que requiere el entorno operativo para representar nombres de ficheros temporales creados por "tmpnam".

NULL
#define NULL <0, 0L, o (void *)0>
Constante de puntero nulo que es utilizable como una expresión de direccionamiento constante.

SEEK_CUR
#define SEEK_CUR <expresion constante entera>
Valor del argumento "modo" para "fseek" para indicar la búsqueda relativa a la posición actual del fichero.

SEEK_END
#define SEEK_END <expresion constante entera>
Valor del argumento "modo" para "fseek" para indicar la búsqueda relativa al final del fichero.

SEEK_SET
#define SEEK_SET <expresion constante entera>
Valor del argumento "modo" para "fseek" para indicar la búsqueda relativa al comienzo del fichero.

TMP_MAX
#define TMP_MAX <expresion constante entera 25>
Número máximo de nombres de ficheros diferentes creados por la función "tmpnam".

clearerr
void clearerr (FILE *flujo);
Limpia los indicadores de error y de fin de fichero para el flujo de datos "flujo".

fclose
int fclose (FILE *flujo);
Cierra el fichero asociado con "flujo". Previamente, escribe lo que queda en el "buffer", descarta cualquier entrada por "buffer" incompleta y libera cualquier "buffer". Devuelve 0 en caso de éxito y EOF en caso contrario.

feof
int feof (FILE *flujo);
Devuelve un valor distinto de 0 si el indicador de final de fichero está a 1.

ferror
int ferror (FILE *flujo);
Devuelve un valor distinto de 0 si el indicador de error está puesto a 1.

fflush
int fflush (FILE *flujo);
Se escibe cualquier salida por "buffer" que esté pendiente. Devuelve 0 si tiene éxito; en caso contrario, devuelve EOF;

fgetc
int fgetc (FILE *flujo);
Lee el siguiente carácter por "flujo", avanza el indicador de posición y devuelve "(int) (unsigned char)". Devuelve EOF si pone a 1 el indicador de fin de fichero o el de error.

fgetpos
int fgetpos (FILE *flujo, fpos_t *pos);
Almacena el indicador de posición de fichero en "pos" y devuelve 0 si tiene éxito; en caso contrario, almacena un valor positivo en "errno" y devuelve un valor distinto de cero.

fgets
char *fgets (char *s, int n, FILE *flujo);
Lee caracteres por "flujo" y los almacena en elementos sucesivos del "array" que comienza en "s", continuando hasta que almacene "n-1" caracteres, almacene un carácter "NL" o ponga a 1 los indicadores de error o fin de fichero. Si almacena un carácter, concluye almacenando un carácter nulo en el siguiente elemento del "array". Devuelve "s" si almacena algún carácter y no ha puesto a 1 el indicador de error; en caso contrario, devuelve un puntero nulo.

fopen
FILE *fopen(const char *nombre_fichero, const char *modo);
Abre el fichero de nombre "nombre_fichero", lo asocia con un flujo de datos y devuelve un puntero al mismo. Si falla, devuelve un puntero nulo.
Los caracteres iniciales de "modo" deben ser alguno de los siguientes:
"r", "w", "a", "rb", "wb", "ab", "r+", "w+", "a+", "rb+", "wb+", "ab+"

"r", para abrir fichero de texto existente para lectura.
"w", para crear fichero de texto o abrir y truncar uno existente, para escritura.
"a", para crear fichero de texto o abrir uno existente, para escritura. El indicador de posición se coloca al final del fichero antes de cada escritura.
"rb", para abrir fichero binario existente para lectura.
"r+", para abrir fichero de texto existente para lectura y escritura.
"rb+", para abrir fichero binario existente para lectura y escritura.

fpos_t
typedef do-tipo fpos_t;
Tipo de datos para contener el valor del indicador de posición del fichero almacenado por "fsetpos" y accedido por "fgetpos".

fprintf
int fprintf (FILE *flujo, const char *formato, ...);
Genera texto formateado, bajo el control del formato "formato" y escribe los caracteres generados por "flujo". Devuelve el número de caracteres generados o un valor negativo en caso de error.

fputc
int fputc (int c, FILE *flujo);
Escribe el carácter "(unsigned char) c" por "flujo", avanza el indicador de posición del fichero y devuelve "(int)(unsigned char) c". En caso de error, devuelve "EOF".

fputs
int fputs (const char *s, FILE *flujo);
Escribe los caracteres de la cadena "s" por "flujo". No escribe el carácter nulo de terminación. En caso de éxito, devuelve un valor no negativo; en caso de error, devuelve "EOF".

fread
size_t fread (void *p, size_t longitud, size_t nelem, FILE *flujo);
Lee caracteres por "flujo" y los almacena en el "array" que comienza en "p" hasta que se almacenen "longitud*nelem" caracteres o se active la condicion de error o la de fin de fichero. Devuelve "n/longitud", siendo "n" el número de caracteres leidos. Si "n" no es múltiplo de "longitud", el valor almacenado en el último elemento queda indeterminado.

freopen
FILE *freopen (const char *nombre_fichero, const char *modo, FILE *flujo);
Cierra el fichero asociado con "flujo" y abre el fichero "nombre_fichero" y lo asocia a "flujo". Devuelve "flujo" si la apertura tiene éxito; en caso contrario, devuelve un puntero nulo.

fscanf
int fscanf (FILE *flujo, const char *formato, ...);
Lee texto y convierte a la representación interna según el formato especificado en "formato". Devuelve el número de entradas emparejadas y asignadas, o "EOF" si no se almacenan valores antes de que se active el indicador de error o de fin de fichero.

fseek
int fseek (FILE *flujo, long desp, int origen);
Activa el indicador de posición de "flujo" según los valores de "desp" y "origen", limpia el indicador de fin de fichero y devuelve 0 si hay éxito.
Valores de "origen": "SEEK_SET" indica el principio de fichero, "SEEK_CUR" indica la posición actual, "SEEK_END" indica el final de fichero.
Para un fichero binario, "desp" es un desplazamiento con signo expresado en número de "bytes", que se añade al indicador de posición indicado por "origen".
Para un fichero de texto, el valor de "desp" puede ser 0 o el valor devuelto por "ftell".

fsetpos
int fsetpos (FILE *flujo, const fpos_t *pos);
Asigna el valor de "pos" al indicador de posición de "flujo", limpia el indicador de fin de fichero y devuelve 0 si ha tenido éxito.

ftell
long ftell (FILE *flujo);
Devuelve una forma codificada del indicador de posición. Para un fichero binario, devuelve el número de "bytes" desde el principio de fichero. Para un fichero de texto, le efecto depende del entorno operativo.
En caso de error, devuelve -1.

fwrite
size_t fwrite(const void *p, size_t longitud, size_t nelem, FILE *flujo);
Escribe caracteres por "flujo", tomandolos a partir de la dirección "p", hasta que se hayan escrito "longitud*nelem" caracteres o se produzca error. Devuelve "n/longitud", siendo "n" el número de caracteres escritos.

getc
int getc (FILE *flujo);
Tiene el mismo efecto que "fgetc".

getchar
int getchar (void);
Tiene el mismo efecto que "fgetc(stdin)".

gets
char *gets (char *s);
Lee caracteres por el flujo estándar de entrada ("stdin") y los almacena en el "array" que comienza en "s" hasta que se almacena un carácter "NL" o se active el indicador de error o el de fin de fichero. Si almacena algún elemento, termina almacenando un carácter nulo. Devuelve "s" si almacena algún carácter. Sustituye el carácter NL por ‘\0’.

perror
void perror (const char *s);
Escribe una línea de texto por "stderr". Escribe la cadena "s", seguida por dos puntos (":") y un espacio. Después escribe la misma cadena que devuelve "strerror(errno)" seguida por "NL".

printf
int printf (const char *formato, ...);
Escribe texto formateado por el flujo "stdout", según las especificaciones de "formato" y la lista de expresiones. Devuelve el número de caracteres escritos o un valor negativo en caso de error.

putc
int putc (int c; FILE *flujo);
Tiene el mismo efecto que "fputc".

putchar
int puchar (int c);
Tiene el mismo efecto que "fputc(c, stdout)".

puts
int puts (const char *s);
Escribe los caracteres de la cadena "s" por el flujo "stdout". Escribe un carácter "NL" en lugar del nulo de terminación. Devuelve un valor no negativo. En caso de error, devuelve EOF.

remove
int remove <9const char *nombre_fichero);
Elimina el fichero "nombre_fichero".

rename
int rename (const char *viejo, const char *nuevo);
Renombra al fichero de nombre "viejo", poniendole el nombre "nuevo".

rewind
void rewind (FILE *flujo)
La función llama a "fseek(flujo, 0L, SEEK_SET)" y limpia el indicador de error para "flujo".

scanf
int scanf (const char *formato, ...);
Lee texto por el flujo "stdin" y lo almacena según las especificaciones de "formato". Devuelve el número de valores asignados o "EOF" si se produce error o se alcanza fin de fichero sin producirse lectura.

setbuf
void setbuf (FILE *flujo, char *buf);
Si "buf" es un puntero nulo, se desactiva el uso de "buffer", en caso contrario llama a "setvbuf(flujo, buf_IOFBF, BUFSIZ)".

setvbuf
int setvbuf (FILE *flujo, char *buf, int modo, size_t longitud);
Establece el uso de "buffer" para "flujo". Debe invocarse después de abrir el fichero y antes de realizar cualquier operación sobre él.
EL "modo" "_IOFBF" es para uso completo de "buffer".
El "modo" "_IOLBF" es para usar "buffer" de linea con ficheros de texto.
EL "modo" "_IONBF" es para no usar "buffer".
Si "buf" no es un puntero nulo , se tomará como la dirección base de un "array" que se usará como "buffer". Si "buf" es nulo, se asigna otro "buffer" que será liberado al cerrar el fichero.

size_t
typedef ui-tipo size_t;
Tipo entero sin signo que se declara para contener el resultado de "sizeof".

sprintf
int sprintf (char *s, const char *format, ...);
Genera texto formateado y lo almacena en un "array" que comienza en "s", poniendo un carácter nulo al final.

sscanf
int sscanf (const char *s, const char *format, ...);
Lee texto formateado tomandolo del "array" que comienza en "s".

stderr
#define stderr <puntero a FILE>
Produce un puntero al flujo de salida estándar para mensajes de error.

stdin
#define stdin <puntero a FILE>
Produce un puntero al flujo de entrada estándar.

stdout
#define stdout <puntero a FILE>
Produce un puntero al flujo de salida estándar.

tmpfile
FILE *tmpfile (void);
Crea un fichero binario temporal que se elimina cuando se cierra o cuando termina la ejecución del programa. Tiene el mismo efecto que abrir con "fopen" y "modo" "wb+".

tmpnam
char *tmpnam (char s[L_tmpnam]);
"tmpnam(NULL)" crea una cadena que sirve como nombre de fichero único y devuelve un puntero a un "array" interno estático. Invocando "tmpnam(s)", la cadena se almacena en "s" y se devuelve su dirección como valor de la función. Se genera un nombre distinto cada vez que se invoca. Se garantiza un número "TMP_MAX" de nombres distintos durante la ejecución del programa.

ungetc
int ungetc (int c, FILE *flujo);
Devuelve "c" (convertido en "unsigned char") al flujo de datos, donde podrá recuperarse en la siguiente lectura.

vfprintf, vprintf, vsprintf
int vfprintf (FILE *flujo, const char *formato, va_list ap);
int vprintf (const char *formato, va_list ap);
int vsprintf (char *s, const char *formato, va_list ap);
Son equivalentes a las correspondientes formas de "printf", con la diferencia de que usan la infomación de contexto designada por "ap" para acceder a los argumentos adicionales. Con "va_start" se inicializa "ap", con "va_arg" se asigna otro valor que es el tipo y valor del siguiente argumento. Hay que llamar a "va_end" después de haber procesado todos los argumentos, pero antes de que termine la función.

<stdlib.h>

Contiene las declaraciones de una colección de funciones útiles y la definición de tipos y macros para usarlas.

EXIT_FAILURE
#define EXIT_FAILURE <expresion entera>
Produce el valor del argumento "estado" para "exit" que informa de una terminación sin éxito.

EXIT_SUCCES
#define EXIT_SUCCES <expresion entera>
Produce el valor del argumento "estado" para "exit" que informa de una terminación con éxito.

MB_CUR_MAX
#define MB_CUR_MAX <expresion entera 1>
Produce el máximo número de caracteres que comprende un carácter multibyte local. Su valor es menor o igual que MB_LEN_MAX.

NULL
#define NULL <0, 0L, o (void *) 0>
Produce una constante de puntero nulo que es utilizable como una expresión de direccionamiento constante.

RAND_MAX
#define RAND_MAX <expresion constante entera 32767>
Produce el máximo valor devuelto por "rand".

abort
void abort (void);
LLama a "raise(SIGABRT)" que produce la señal de abortar, lo que causa la terminación anormal del programa informando al entorno operativo.

abs
int abs (int i);
Devuelve el valor absoluto de "i".

atexit
int atexit (void (*func) (void));
Registra la función cuya dirección es "func" para ser llamada por "exit". Se pueden registrar al menos 32 funciones. La función "exit" llama a las funciones en orden inverso de registro.

atof
double atof (const char *s);
Convierte los caracteres de la cadena "s" a la representación interna de tipo "double" y devuelve ese valor. Es semejante a "strtod(s,NULL)", pero no almacena necesariamente código de error en "errno" si ocurre error de conversión.

atoi
int atoi (const char *s);
Convierte los caracteres de la cadena "s" a la representación interna de tipo "int" y devuelve ese valor. Es semejante a "(int)strtol(s,NULL,10)", pero no se almacena código de error en "errno".

atol
long atol (const char *s);
Convierte los caracteres de la cadena "s" a la representación interna de tipo "long" y devuelve ese valor. Es semejante a "strtol(s,NULL,10)", pero no almacena código de error en "errno".

bsearch
void *bsearch (const void *clave, const void *base, size_t nelem,
size_t longitud,
int(*cmp) (const void *ckey, const void *celem));
Busca en eun "array" de valores ordenados en sentido creciente y devuelve la dirección de un elemento del "array" que es igual a la clave de búsqueda "clave". Si no existe ninguno, devuelve un puntero nulo. El "array" consiste en "nelem" elementos, de tamaño "longitud", expresado en "bytes", empezando en la dirección "base".
LLama a la función de dirección "cmp" para comparar la clave de búsqueda con los elementos del "array". La función debe devolver un valor negativo si la clave "ckey" es menor que el elemento "celem", cero si son iguales y un valor positivo si la clave es mayor.

calloc
void *calloc (size_t nelem, size_t longitud);
Asigna una localización en memoria a un objeto de datos "array" que contiene "nelem" elementos de tamaño "longitud", asigna ceros a todos los "bytes" del "array" y devuelve la dirección del primer elemento en caso de éxito; en caso contrario, devuelve un puntero nulo.

div
div_t div (int numer, int denom);
Divide "numer" entre "denom" y devuelve el cociente y el resto en una estructura de tipo "div_t". El miembro "coc" es el cociente truncado hacia cero, el miembro "res" es el resto .

div_t
```
typedef struct {
   int coc;   /* cociente */
   int res;   /* resto */
} div_t;
```
Es un tipo que se declara para contener el valor devuelto por la función "div". La estructura contiene miembros que representan el cociente y el resto de una división entera con signo entre operandos de tipo "int".

exit
void exit (int status);
LLama a todas las funciones registradas por "atexit", cierra todos los ficheros y devuelve el control al entorno operativo.

free
void free (void *p);
Si "p" no es un puntero nulo, la función libera la memoria asignada al objeto de datos cuya dirección es "p"; en caso contrario, no hace nada. Se puede liberar la memoria asignada con "calloc", "malloc", "realloc".

getenv
char *getenv (const char *nombre);
Devuelve el valor de la variable de entorno identificada por "nombre".

labs
long labs (long i);
Devuelve el valor absoluto de "i".

ldiv
ldiv_t ldiv (long numer, long denom);
Es semejante a "div", pero aplicada a valores de tipo "long".

ldiv_t
typedef struct {
    long coc;   /* cociente */
    long res;   /* resto */
} ldiv_t;
Es un tipo estructura declarado para contener el valor que devuelve "ldiv".

malloc
void *malloc (size_t longitud);
Asigna una dirección de memoria para un objeto de datos de tamaño "longitud" y devuelve esa dirección.

mblen
int mblen (const char *s, size_t n);
Si "s" no es un puntero nulo, devuelve el número de caracteres en la cadena multibyte "s" que constituyen el siguiente carácter multibyte, o devuelve -1 si los siguientes n (o los restantes caracteres) no comprenden un carácter multibyte válido.

mbstowcs
size_t mbstowcs (wchar_t *wcs, const char *s, size_t n);
Almacena una cadena de caractes amplios en elementos del "array" que empieza en "wcs", convirtiendo, en orden, cada uno de los caracteres multibyte de la cadena multibyte "s".

mbtowc
int mbtowc (wchar_t *pwc, const char *s, size_t n);
Si "s" no es un puntero nulo, la función determina el número de caracteres en la cadena multibyte "s" que constituyen el siguiente carácter multibyte. Si "pwc" no es un puntero nulo, la función convierte el siguiente carácter multibyte en su valor correspondiente de carácter amplio y almacena este valor en "*pwc".

qsort
void qsort (void *base, size_t nelem, size_t longitud,
        int (*cmp) (const void *e1, const void *e2));
Ordena un "array" que comienza en "base", compuesto por "nelem" elementos, cada uno de los cuales tiene un tamaño de "longitud" expresado en "bytes".
LLama a la función de comparación cuya dirección es "cmp", que debe devolver un valor negativo si "e1" es menor que "e2", 0 si son iguales y positivo si "e1" es mayor que "e2".

rand
int rand(void);
Calcula un número pseudoaleatorio "x" basado en un valor inicial (semilla) almacenado en un objeto de datos interno de duración estática, altera el valor almacenado y devuelve "x". El valor devuelto está comprendido en el intervalo [0,RAND_MAX].

realloc
void *realloc (void *p, size_t longitud);
Cambia el tamaño de la memoria apuntada por "p" al que se indica con "longitud". Asigna una dirección de memoria para un objeto de datos de tamaño "longitud", copiando los valores almacenados en "p". Devuelve la nueva dirección de memoria asignada.

size_t
typedef ui-tipo size_t;
Es el tipo entero sin signo que se declara para contener el resultado del operador "sizeof".

srand
void srand (unsigned semilla);
Utiliza "semilla", en un objeto de datos de duración estática, para generar una secuencia de números pseudoaleatorios con "rand".

strtod
double strtod (const char *s, char **finptr);
Convierte los caracteres iniciales de una cadena "s" en la correspondiente representación interna de tipo "double" y devuelve ese valor. Si "finptr" no es un puntero nulo, la función almacena en él un puntero al resto de la cadena que no se ha convertido.

strtol
long strtol (const char *s, char **finptr, int base);
Convierte los caracteres iniciales de una caena "s" en la correspondiente representación interna de tipo "long" y devuelve ese valor. SI "finptr" no es un puntero nulo, la función almacena en él un puntero al resto de la cadena que no se ha convertido.

strtoul
unsigned strtoul (const char *s, char **finptr, int base);
Es semejante a "strtol", pero aplicada al tipo "unsigned long".

system
int system (const char *s);
Si "s" no es un puntero nulo, se pasa la cadena "s" para que sea ejecutada por el intérprete de comandos del entorno operativo, y devuelve la información de estado proporcionada por el intérprete.
Si "s" es un puntero nulo, la función devuelve un valor distinto de cero si existe intérprete de comandos en el entorno operativo.

wchar_t
typedef i-tipo wchar_t;
Es el tipo entero de la constante de caracteres amplios L'X'. Se declara un objeto de datos de tipo "wchar_t" para almacenar un carácter amplio.

wcstombs
size_t wcstombs (char *s, const wchar_t *wcs, size_t n);
Almacena una cadena multibyte en sucesivos elementos del "array" que comienza en "s", convirtiendo, en orden, cada uno de los caracteres amplios en la cadena "wcs".

wctomb
int wctomb (char *s, wchar_t wchar);
Si "s" no es un puntero nulo, la función determina y devuelve el número de caracteres que se necesitan para representar el carácter multibyte correspondiente al carácter amplio "wchar". La función convierte "wchar" en su correspondiente carácter multibyte almacenándolo en el "array s".

<string.h>

Contiene la declaración de una colección de funciones útiles para manejar cadenas y otros arrays de caracteres.

NULL
#define NULL <0, 0L, o (void *)0>
Produce una constante de puntero nulo que es utilizable como una expresión de direccionamiento constante.

memchr
void *memchr (const void *s, int c, size_t n);
Busca el primer elemento de un "array de unsigned char" que es igual a "(unsigned char) c". El "array" comienza en "s" y tiene "n" elementos.
En caso de éxito, devuelve la dirección de elemento buscado; en caso contrario, devuelve un puntero nulo.

memcmp
int memcmp (const void *s1, const void *s2, size_t n);
Compara los elementos de dos "arrays de unsined char", con direcciones base "s1" y "s2", y "n" elementos, hasta que encuentra elementos diferentes. Si todos los elementos son iguales, devuelve 0. Si el elemento diferente de "s1" es mayor que el de "s2", devuelve un valor positivo; en caso contrario, devuelve un valor negativo.

memcpy
void *memcpy (void *s1, const void *s2, size_t n);
Copia el "array de char" que empieza en "s2" en el "array de char" que empieza en "s1". Devuelve "s1".

memmove
void *memmove (void *s1, const void *s2, size_t n);
Tiene el mismo efecto que "memcpy", pero actúa incluso si los "arrays" se solapan. En este caso, se accede a cada valor de "s2" antes de almacenar un nuevo valor en ese elemento.

memset
void *memset (void *s, int c, size_t n);
Almacena "(unsigned char) c" en cada uno de los elementos del "array de unsigned char" que empieza en "s" y tiene "n" elementos. Devuelve "s".

size_t
typedef ui-tipo size_t;
Es el tipo entero sin signo de un objeto de datos declarado para contener el resultado del operador "sizeof".

strcat
char *strcat (char *s1, const char *s2);
Copia la cadena "s2", incluyendo el nulo de terminación, en elementos sucesivos del "array de char" que almacena la cadena "s1", empezando en el elemento que almacena el nulo de terminación de "s1". Devuelve "s1".

strchr
char *strchr (const char *s, int c);
Busca el primer elemento de la cadena "s" que sea igual a "(char)c". Considera el nulo de terminación como parte de la cadena. En caso de éxito, devuelve la dirección del elemento emparejado; en caso contrario, devuelve un puntero nulo.

strcmp
int strcmp (const char *s1, const char *s2);
Compara los elementos de dos cadenas "s1" y "s2" hasta que encuentra elementos diferentes. Si todos son iguales, devuelve 0. Si el elemento diferente de "s1" es mayor que el de "s2", devuelve un valor mayor que cero; en caso contrario, devuelve un valor menor que cero.

strcoll
int strcoll (const char *s1, const char *s2);
Compara dos cadenas "s1" y "s2" utilizando una secuencia de ordenación especial diferente de la ASCII.

strcpy
char *strcpy (char *s1, const char *s2);
Copia la cadena "s2", incluyendo el nulo, en el "array de char" que comienza en "s1". Devuelve "s1".

strcspn
size_t strcspn (const char *s1, const char *s2);
Busca el primer elemento "s1[i]" de la cadena "s1" que sea igual a cualquiera de los elementos de la cadena "s2" y devuelve "i".

strerror
char *strerror (int codigo_error);
Devuelve un puntero a un objeto de datos interno de duración estática que contiene el mensaje correspondiente al código de error "codigo_error".

strlen
size_t strlen (const char *s);
Devuelve el número de caracteres de la cadena "s", sin incluir el nulo de terminación.

strncat
char *strncat (char *s1, const char *s2, size_t n);
Copia "n" elementos de la cadena "s2" en la cadena "s1" a partir de su nulo de terminación. Al final pone el nulo de terminación y devuelve "s1".

strncmp
int strncmp (const char *s1, const char *s2, size_t n);
Compara los elementos de las cadenas "s1" y "s2" hasta que encuentra alguno diferente o hasta que se han comparado "n" elementos. Si todos los elementos son iguales, devuelve 0. Si el elemento diferente de "s1" es mayor que el de "s2" (tomados como "unsigned char"), devuelve un número positivo. En caso contrario, devuelve un número negativo.

strncpy
char *strncpy (char *s1, const char *s2, size_t n);
Copia la cadena "s2", sin incluir el nulo, en la cadena "s1". Copia no más de "n" caracteres de "s2". Entonces almacena, cero o más caracteres nulos si son necesarios para completar un total de "n" caracteres. Devuelve "s1".

strpbrk
char *strpbrk (const char *s1, const char *s2);
Busca el primer elemento "s1[i]" en la cadena "s1" que sea igual a cualquiera de los elementos de "s2". Si "s1[i]" no es el nulo de terminación, devuelve "&s1[i]"; en caso contrario, devuelve un puntero nulo.

strrchr
char *strrchr (const char *s, int c);
Busca el último elemento de la cadena "s" que es igual a "(char)c". En caso de éxito, devuelve la dirección de tal elemento; en caso contrario, devuelve un puntero nulo.

strspn
size_t strspn (const char *s1, const char *s2);
Busca el primer elemento "s1[i]" en la cadena "s1" que no sea igual a ninguno de los elementos de "s2" y devuelve "i".

strstr
char *strstr (const char *s1, const char *s2);
Busca la primera secuencia de elementos en la cadena "s1" que se empareje con los elementos de la cadena "s2", sin incluir su nulo. En caso de éxito, devuelve la dirección del primer elemento emparejado; en caso contrario, devuelve un puntero nulo.

strtok
char *strtok (char *s1, const char *s2);
Permite separar una cadena "s1" en partes usando como delimitadores los caractes de otra "s2". Cada vez que se invoca, devuelve un puntero a la siguiente palabra de la cadena "s1". Devuelve un puntero nulo cuando no existe ninguna palabra que devolver.
La primera vez que se llama a "strtok", se utiliza realmente "s1" en la llamada. Las llamadas posteriores utilizan un puntero nulo como primer argumento. Se puede usar un conjunto diferente de delimitadores en cada llamada.
Esta función modifica la cadena "s1". Cada vez que se encuentra una palabra, se pone un carácter nulo donde estaba el delimitador.

strxfrm
size_t strxfrm (char *s1, const char *s2, size_t n);
Se usa en entornos de lenguajes extranjeros que no utilicen el secuenciamiento ASCII.
Transforma los "n" primeros elementos de la cadena "s2" para que pueda usarse con la función "strcmp". La función "strxfrm" coloca entonces el resultado en la cadena "s1". Después de la transformación, el resultado de "strcmp" con "s1" y de "strcoll" con la cadena original "s2" será el mismo. La función devuelve la longitud de la cadena transformada.

<time.h>

Contiene la declaración de algunas funciones para manejar fechas. Las funciones comparten dos objetos de datos de duración estática, una cadena de tiempo de tipo "array de char" y una estructura de tiempo de tipo "struct tm".

CLOCKS_PER_SECOND
#define CLOCKS_PER_SECOND <valor_aritmetico>
Produce el número de pulsos de reloj, devuelto por "clock" en un segundo.

NULL
#define NULL <0, 0L, o (void *)0>
Produce una constante de puntero nulo que se puede utilizar como una expresión constante de direccionamiento.

asctime
char *asctime (const struct tm *tp);
Convierte la información almacenada en la estructura apuntada por "tp" en una cadena de caracteres que expresa la fecha y hora en lengua inglesa con la forma: Mon Apr 12 09:12:05 1993\n\0. Devuelve la dirección de la cadena.
El puntero que se pasa, "tp", puede obtenerse con "localtime" o con "gmtime".

clock
clock_t clock (void);
Devuelve el número de pulsaciones de reloj de un lapso de tiempo del procesador, contando desde el momento de arranque del programa. Devuelve "-1" si el entorno operativo no puede hacer esa medida.

clock_t
typedef a-tipo clock_t;
Es el tipo aritmético "a-tipo" de un objeto de datos que se declara para contener el valor que devuelve "clock". El valor representa el lapso de tiempo del procesdor.

ctime
char *ctime (const time_t *cal);
Convierte el tiempo de calendario que está en "*cal" a una representación de texto de la hora local. Es equivalente a "asctime(localtime(cal))". La hora de calendario se obtiene normalmente con una llamada a "time".

difftime
double difftime (time_t t1, time_t t0);
Devuelve la diferencia, en segundos, entre los tiempos de calendario "t0" y "t1".

gmtime
struct tm *gmtime (const time_t *tod);
Almacena en la estructura de tiempo una codificación del tiempo de calendario apuntado palmacenado en "*tod", expresado en Tiempo Universal Coordinado (UTC, antes GMT). Devuelve la dirección a la estructura de tiempo.
El valor de "*tod" se obtiene normalmente llamando a "time".

localtime
struct tm *gmtime (const time_t *tod);
Almacena en la estructura de tiempo una codificación del tiempo de calendario almacenado en "*tod", expresado como hora local. Devuelve la dirección de la estructura. El valor de "*tod" se obtiene normalmente llmando a "time".

mktime
time_t mktime (struct tm *tp);
Devuelve la hora de calendario,con la representación propia de "time", correspondiente a la hora local almacenada en la estructura apuntada por "tp". Devuelve "-1" si la información no corresponde a una hora de calendario válida.

size_t
typedef ui-tipo size_t;
Es el tipo entero sin signo de un objeto de datos que se declara para contener el resultado del operador "sizeof".

strftime
size_t strftime (char *s, size_t n, const char *formato,
const struct tm *tp);
Toma los valores de la estructura de tiempo apuntada por "tp" y genera texto formateado, según las especificaciones de "formato", almacenandolo en la cadena "s" de tamaño "n". Hay 22 especificaciones de formato.

time
time_t time (time_t *tod);
Devuelve la hora actual de calendario del sistema. Si el entorno operativo no puede determinarla, devuelve "-1".
Si "tod" no es un puntero nulo, la hora también queda asignada a "*tod".

time_t
typedef a-tipo time_t;
Es el tipo aritmético de un objeto de datos que se declara para almacenar el valor devuelto por "time". El valor representa la hora de calendario.

tm
```
struct tm {
  int tm_sec;    /*  segundos después del minuto, 0-59  */
  int tm_min;    /*  minutos después de la hora, 0-59  */
  int tm_hour;   /*  hora del día, 0-23  */
  int tm_mday;   /*  día del mes, 1-31  */
  int tm_mon;    /*  mes del año, 0-11  */
  int tm_year;   /*  años desde 1900  */
  int tm_wday;   /*  días desde el domingo, 0-6  */
  int tm_yday;   /*  día del año, 0-365  */
  int tm_isdst;  /*  indicador de ajuste horario */
  };
```
Se utiliza para mantener la fecha y hora separadas en sus compoentes.
El miembro "tm_isdst" tiene un valor positivo si el ejuste horario es efectivo, tiene el valor "0" si no es efectivo o un valor negativo si el entorno operativo no puede determinar su estado.