Tabelle riepilogative della libreria C e POSIX

va_start() 1359 vfprintf() 1421 vfscanf() 1428
vprintf() 1421 vscanf() 1428 vsnprintf() 1421
vsprintf() 1421 vsscanf() 1428 WCHAR_MAX 1361
WCHAR_MIN 1361 wchar_t 1361 wcstombs() 1379
wctomb() 1379 WINT_MAX 1361 WINT_MIN 1361 wint_t
1361 _Exit() 1379 _IOFBF 1406 _IOLBF 1406 _IONBF 1406

File «stdarg.h»

Macroistruzione	Descrizione
`void va_start (va_list` ***ap***`,` `  `***parametro_n***`);`	Inizializza la variabile ***ap***, di tipo '**`va_list`**', in modo che punti all'area di memoria immediatamente successiva al parametro indicato, il quale deve essere l'ultimo.
tipo `va_arg (va_list` ***ap***`,` ***tipo***`);`	Restituisce il contenuto dell'area di memoria a cui punta ***ap***, utilizzando il tipo indicato, incrementando contestualmente il puntatore in modo che, al termine, si trovi nell'area di memoria immediatamente successiva.
`void va_copy (va_list` ***dst***`,` `  va_list` ***org***`);`	Copia il puntatore ***org*** nella variabile ***dst***.
`void va_end (va_list` ***ap***`);`	Conclude l'utilizzo del puntatore ***ap***.

«

Macro-variabile	Descrizione
`CHAR_BIT`	Quantità di bit utilizzata per rappresentare il tipo '**`char`**', con o senza segno. In altri termini è l'unità di memorizzazione più piccola con cui si può gestire l'insieme di caratteri minimo. Di norma si tratta di 8 bit.
`SCHAR_MIN` `SCHAR_MAX`	Il valore minimo e il valore massimo rappresentabile in una variabile '**`signed char`**'.
`UCHAR_MAX`	Il valore massimo rappresentabile in una variabile '**`unsigned char`**'. Il valore minimo è zero.
`CHAR_MIN` `CHAR_MAX`	Il valore minimo e il valore massimo rappresentabile in una variabile '**`char`**'. Questi valori dipendono dal fatto che il tipo '**`char`**' sia da intendere equivalente a un tipo '**`unsigned char`**' o '**`signed char`**', da cui ereditano i limiti.
`MB_LEN_MAX`	La quantità massima di byte che possono essere usati per rappresentare un carattere multibyte, qualunque sia la configurazione locale.
`SHRT_MIN` `SHRT_MAX`	Il valore minimo e il valore massimo rappresentabile in una variabile '**`short int`**'.
`USHRT_MAX`	Il valore massimo rappresentabile in una variabile '**`unsigned short int`**'. Il valore minimo è zero.
`INT_MIN` `INT_MAX`	Il valore minimo e il valore massimo rappresentabile in una variabile '**`int`**'.
`UINT_MAX`	Il valore massimo rappresentabile in una variabile '**`unsigned int`**'. Il valore minimo è zero.

Macro-variabile	Descrizione

`LONG_MIN`
 	
`LONG_MAX`
	Il valore minimo e il valore massimo rappresentabile in una variabile '**`long int`**'.

`ULONG_MAX`	Il valore massimo rappresentabile in una variabile '**`unsigned long int`**'. Il valore minimo è zero.

`LLONG_MIN`
 	
`LLONG_MAX`
	Il valore minimo e il valore massimo rappresentabile in una variabile '**`long long int`**'.

`ULLONG_MAX`	Il valore massimo rappresentabile in una variabile '**`unsigned long long int`**'. Il valore minimo è zero.

File «stdint.h»

Con segno	Senza segno	Descrizione

`int8_t`
 	
`int16_t`
 	
`int32_t`
 	
`int64_t`
	`uint8_t`
	
	`uint16_t`
	
	`uint32_t`
	
	`uint64_t`
		Tipo intero, facoltativo, il cui rango è prestabilito esattamente.

Con segno	Senza segno	Descrizione

`INT8_MIN`
`INT8_MAX`
`INT16_MIN`
`INT16_MAX`
`INT32_MIN`
`INT32_MAX`
`INT64_MIN`
`INT64_MAX`
	`UINT8_MAX`
	`UINT16_MAX`
	`UINT32_MAX`
	`UINT64_MAX`
		Limiti minimi e massimi dei tipi '**`int`*n*`_t`**' e '**`uint`*n*`_t`**'.

`int_least8_t`
`int_least16_t`
`int_least32_t`
`int_least64_t`
	`uint_least8_t`
	`uint_least16_t`
	`uint_least32_t`
	`uint_least64_t`
		Tipo intero con un rango minimo stabilito.

Con segno	Senza segno	Descrizione

`INT_LEAST8_MIN`
`INT_LEAST8_MAX`
`INT_LEAST16_MIN`
`INT_LEAST16_MAX`
`INT_LEAST32_MIN`
`INT_LEAST32_MAX`
`INT_LEAST64_MIN`
`INT_LEAST64_MAX`
	`UINT_LEAST8_MAX`
	`UINT_LEAST16_MAX`
	`UINT_LEAST32_MAX`
	`UINT_LEAST64_MAX`
		Limiti minimi e massimi dei tipi '**`int_least`*n*`_t`**' e '**`uint_least`*n*`_t`**'.

`INT8_C(`***val***`)`
`INT16_C(`***val***`)`
`INT32_C(`***val***`)`
`INT64_C(`***val***`)`
	`UINT8_C(`***val***`)`
	`UINT16_C(`***val***`)`
	`UINT32_C(`***val***`)`
	`UINT64_C(`***val***`)`
		Macroistruzione per attribuire l'estensione che definisce il tipo corretto a un valore costante, da intendere secondo il tipo '**`int_least`*n*`_t`**' o '**`uint_least`*n*`_t`**'.

`int_fast8_t`
`int_fast16_t`
`int_fast32_t`
`int_fast64_t`
	`uint_fast8_t`
	`uint_fast16_t`
	`uint_fast32_t`
	`uint_fast64_t`
		Tipo intero con un rango minimo stabilito, con caratteristiche ottimali per la velocità elaborativa.

Con segno	Senza segno	Descrizione

`INT_FAST8_MIN`
`INT_FAST8_MAX`
`INT_FAST16_MIN`
`INT_FAST16_MAX`
`INT_FAST32_MIN`
`INT_FAST32_MAX`
`INT_FAST64_MIN`
`INT_FAST64_MAX`
	`UINT_FAST8_MAX`
	`UINT_FAST16_MAX`
	`UINT_FAST32_MAX`
	`UINT_FAST64_MAX`
		Limiti minimi e massimi dei tipi '**`int_fast`*n*`_t`**' e '**`uint_fast`*n*`_t`**'.

`intptr_t`	`uintptr_t`	Tipo facoltativo intero capace di contenere il valore di un puntatore, convertibile da e verso '**`void *`**'.
`INTPTR_MIN` `INTPTR_MAX`	`UINTPTR_MAX`	Limiti minimi e massimi dei tipi '**`intptr_t`**' e '**`uintptr_t`**'.
`intmax_t`	`uintmax_t`	Tipo intero di rango massimo.
`INTMAX_MIN` `INTMAX_MAX`	`UINTMAX_MAX`	Limiti minimi e massimi dei tipi '**`intmax_t`**' e '**`uintmax_t`**'.

Con segno	Senza segno	Descrizione
`INTMAX_C(`***val***`)`	`UINTMAX_C(`***val***`)`	Macroistruzione per attribuire l'estensione che definisce il tipo corretto a un valore costante, da intendere secondo il tipo '**`intmax_t`**' o '**`uintmax_t`**'.
`PTRDIFF_MIN` `PTRDIFF_MAX`		Limiti minimi e massimi del tipo '**`ptrdiff_t`**'.
`SIG_ATOMIC_MIN` `SIG_ATOMIC_MAX`		Limiti minimi e massimi del tipo '**`sig_atomic_t`**'.
	`SIZE_MAX`	Limite massimo del tipo '**`size_t`**' (senza segno).
`WCHAR_MIN` `WCHAR_MAX`		Limiti minimi e massimi del tipo '**`wchar_t`**'.
`WINT_MIN` `WINT_MAX`		Limiti minimi e massimi del tipo '**`wint_t`**'.

File «inttypes.h»

Macro-variabili per la composizione dell'output	Macro-variabili per l'interpretazione dell'input	Esempi schematici

PRId8
PRId16
PRId32
PRId64
	SCNd8
	SCNd16
	SCNd32
	SCNd64
		`int32_t i = INT32_MAX;`
		`…`
		`printf ("i = %010" PRId32 "\n", i);`
		`…`
		`scanf ("%" SCNd32, &i);`

PRIi8
PRIi16
PRIi32
PRIi64
	SCNi8
	SCNi16
	SCNi32
	SCNi64
		`int32_t i = INT32_MAX;`
		`…`
		`printf ("i = %010" PRIi32 "\n", i);`
		`…`
		`scanf ("%" SCNi32, &i);`

PRIdLEAST8
PRIdLEAST16
PRIdLEAST32
PRIdLEAST64
	SCNdLEAST8
	SCNdLEAST16
	SCNdLEAST32
	SCNdLEAST64
		`int_least32_t i = INT_LEAST32_MAX;`
		`…`
		`printf ("i = %010" PRIdLEAST32 "\n", i);`
		`…`
		`scanf ("%" SCNdLEAST32, &i);`

Macro-variabili per la composizione dell'output	Macro-variabili per l'interpretazione dell'input	Esempi schematici

PRIiLEAST8
PRIiLEAST16
PRIiLEAST32
PRIiLEAST64
	SCNiLEAST8
	SCNiLEAST16
	SCNiLEAST32
	SCNiLEAST64
		`int_least32_t i = INT_LEAST32_MAX;`
		`…`
		`printf ("i = %010" PRIiLEAST32 "\n", i);`
		`…`
		`scanf ("%" SCNiLEAST32, &i);`

PRIdFAST8
PRIdFAST16
PRIdFAST32
PRIdFAST64
	SCNdFAST8
	SCNdFAST16
	SCNdFAST32
	SCNdFAST64
		`int_fast32_t i = INT_FAST32_MAX;`
		`…`
		`printf ("i = %010" PRIdFAST32 "\n", i);`
		`…`
		`scanf ("%" SCNdFAST32, &i);`

PRIiFAST8
PRIiFAST16
PRIiFAST32
PRIiFAST64
	SCNiFAST8
	SCNiFAST16
	SCNiFAST32
	SCNiFAST64
		`int_fast32_t i = INT_FAST32_MAX;`
		`…`
		`printf ("i = %010" PRIiFAST32 "\n", i);`
		`…`
		`scanf ("%" SCNiFAST32, &i);`

Macro-variabili per la composizione dell'output	Macro-variabili per l'interpretazione dell'input	Esempi schematici

PRIdMAX
PRIdPTR
PRIiMAX
PRIiPTR
	SCNdMAX
	SCNdPTR
	SCNiMAX
	SCNiPTR
		`intmax_t i = INTMAX_MAX;`
		`…`
		`printf ("i = %020" PRIiMAX "\n", i);`
		`…`
		`scanf ("%" SCNiMAX, &i);`

PRIo8
PRIo16
PRIo32
PRIo64
	SCNo8
	SCNo16
	SCNo32
	SCNo64
		`uint32_t i = UINT32_MAX;`
		`…`
		`printf ("i = %011" PRIo32 "\n", i);`
		`…`
		`scanf ("%" SCNo32, &i);`

PRIoLEAST8
PRIoLEAST16
PRIoLEAST32
PRIoLEAST64
	SCNoLEAST8
	SCNoLEAST16
	SCNoLEAST32
	SCNoLEAST64
		`uint_least32_t i = UINT_LEAST32_MAX;`
		`…`
		`printf ("i = %011" PRIoLEAST32 "\n", i);`
		`…`
		`scanf ("%" SCNoLEAST32, &i);`

Macro-variabili per la composizione dell'output	Macro-variabili per l'interpretazione dell'input	Esempi schematici

PRIoFAST8
PRIoFAST16
PRIoFAST32
PRIoFAST64
	SCNoFAST8
	SCNoFAST16
	SCNoFAST32
	SCNoFAST64
		`uint_fast32_t i = UINT_FAST32_MAX;`
		`…`
		`printf ("i = %011" PRIoFAST32 "\n", i);`
		`…`
		`scanf ("%" SCNoFAST32, &i);`

PRIoMAX
PRIoPTR
	SCNoMAX
	SCNoPTR
		`uintmax_t i = INTMAX_MAX;`
		`…`
		`printf ("i = %022" PRIoMAX "\n", i);`
		`…`
		`scanf ("%" SCNoMAX, &i);`

PRIu8
PRIu16
PRIu32
PRIu64
	SCNu8
	SCNu16
	SCNu32
	SCNu64
		`uint32_t i = UINT32_MAX;`
		`…`
		`printf ("i = %010" PRIu32 "\n", i);`
		`…`
		`scanf ("%" SCNu32, &i);`

PRIuLEAST8
PRIuLEAST16
PRIuLEAST32
PRIuLEAST64
	SCNuLEAST8
	SCNuLEAST16
	SCNuLEAST32
	SCNuLEAST64
		`uint_least32_t i = UINT_LEAST32_MAX;`
		`…`
		`printf ("i = %010" PRIuLEAST32 "\n", i);`
		`…`
		`scanf ("%" SCNuLEAST32, &i);`

Macro-variabili per la composizione dell'output	Macro-variabili per l'interpretazione dell'input	Esempi schematici

PRIuFAST8
PRIuFAST16
PRIuFAST32
PRIuFAST64
	SCNuFAST8
	SCNuFAST16
	SCNuFAST32
	SCNuFAST64
		`uint_fast32_t i = UINT_FAST32_MAX;`
		`…`
		`printf ("i = %010" PRIuFAST32 "\n", i);`
		`…`
		`scanf ("%" SCNuFAST32, &i);`

PRIuMAX
PRIuPTR
	SCNuMAX
	SCNuPTR
		`uintmax_t i = INTMAX_MAX;`
		`…`
		`printf ("i = %022" PRIuMAX "\n", i);`
		`…`
		`scanf ("%" SCNuMAX, &i);`

PRIx8
PRIX8
PRIx16
PRIX16
PRIx32
PRIX32
PRIx64
PRIX64
	SCNx8
	SCNx16
	SCNx32
	SCNx64
		`uint32_t i = UINT32_MAX;`
		`…`
		`printf ("i = %08" PRIx32 "\n", i);`
		`…`
		`scanf ("%" SCNx32, &i);`

Macro-variabili per la composizione dell'output	Macro-variabili per l'interpretazione dell'input	Esempi schematici

PRIxLEAST8
PRIXLEAST8
PRIxLEAST16
PRIXLEAST16
PRIxLEAST32
PRIXLEAST32
PRIxLEAST64
PRIXLEAST64
	SCNxLEAST8
	SCNxLEAST16
	SCNxLEAST32
	SCNxLEAST64
		`uint_least32_t i = UINT_LEAST32_MAX;`
		`…`
		`printf ("i = %08" PRIxLEAST32 "\n", i);`
		`…`
		`scanf ("%" SCNxLEAST32, &i);`

PRIxFAST8
PRIXFAST8
PRIxFAST16
PRIXFAST16
PRIxFAST32
PRIXFAST32
PRIxFAST64
PRIXFAST64
	SCNxFAST8
	SCNxFAST16
	SCNxFAST32
	SCNxFAST64
		`uint_fast32_t i = UINT_FAST32_MAX;`
		`…`
		`printf ("i = %08" PRIxFAST32 "\n", i);`
		`…`
		`scanf ("%" SCNxFAST32, &i);`

PRIxMAX
PRIXMAX
PRIxPTR
PRIXPTR
	SCNxMAX
	SCNxPTR
		`uintmax_t i = UINTMAX_MAX;`
		`…`
		`printf ("i = %016" PRIxMAX "\n", i);`
		`…`
		`scanf ("%" SCNxMAX, &i);`

Funzione	Descrizione
`intmax_t imaxabs (intmax_t` ***j*** `);`	Restituisce il valore assoluto del numero passato come argomento.
`imaxdiv_t imaxdiv (intmax_t` ***numer*** `,` `intmax_t` ***denom*** `);`	Restituisce il risultato della divisione dei due argomenti, in una struttura contenente il risultato intero e il resto della divisione.

Funzione	Descrizione
`intmax_t strtoimax (const char *restrict s, char **restrict p, int base);` `uintmax_t strtouimax (const char *restrict s, char **restrict p, int base);`	Converte la stringa fornita come primo argomento in un numero intero, come si vede dal modello sintattico, interpretando la stringa come numero espresso nella base rappresentata dal parametro ***base***. La conversione avviene fino a dove è possibile riconoscere caratteri che compongono un valore valido; se il secondo argomento è un puntatore a un puntatore valido (un puntatore a un'area di memoria che può contenere a sua volta un puntatore dal tipo '**`char`**'), al suo interno viene memorizzato l'indirizzo finale della scansione, a partire dal quale si trovano caratteri non decifrabili, oppure dove si trova il carattere nullo di terminazione della stringa.

Funzione	Descrizione
`intmax_t wcstoimax (const wchar_t *restrict wcs, wchar_t **restrict p, int base);` `uintmax_t wcstouimax (const wchar_t *restrict wcs, wchar_t **restrict p, int base);`	Converte la stringa estesa fornita come primo argomento in un numero intero, come si vede dal modello sintattico, interpretando la stringa estesa come numero espresso nella base rappresentata dal parametro ***base***. La conversione avviene fino a dove è possibile riconoscere caratteri estesi che compongono un valore valido; se il secondo argomento è un puntatore a un puntatore valido (un puntatore a un'area di memoria che può contenere a sua volta un puntatore dal tipo '**`wchar_t`**'), al suo interno viene memorizzato l'indirizzo finale della scansione, a partire dal quale si trovano caratteri estesi non decifrabili, oppure dove si trova il carattere nullo di terminazione della stringa.

Funzione	Descrizione
`int isalnum (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere alfabetico o numerico. Equivale alla corrispondenza con ***isalpha()*** o con ***isdigit()***.
`int isalpha (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere alfabetico. Equivale alla corrispondenza con ***isupper()*** o con ***islower()***.
`int isblank (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere per la spaziatura orizzontale delle parole.

Funzione	Descrizione
`int iscntrl (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere di controllo.
`int isdigit (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere che rappresenta una cifra decimale.
`int isgraph (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere che ha una rappresentazione grafica, escluso lo spazio.
`int islower (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere alfabetico minuscolo.

Funzione	Descrizione
`int isprint (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere che ha una rappresentazione grafica, incluso lo spazio.
`int ispunct (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere di punteggiatura.
`int isspace (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere di spaziatura, sia orizzontale, sia verticale, incluso il salto pagina e il ritorno a carrello.

Funzione	Descrizione
`int isupper (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere alfabetico maiuscolo.
`int isxdigit (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce un valore diverso da zero se l'argomento corrisponde a un carattere che rappresenta una cifra esadecimale (espressa indifferentemente con lettere minuscole o maiuscole).
`int tolower (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce il carattere fornito come argomento, dopo la conversione in minuscolo, ammesso che ci possa essere una conversione.

Funzione	Descrizione
`int toupper (int c);`	L'argomento rappresentato dal parametro ***c*** è un carattere senza segno convertito in un intero, oppure l'equivalente di '**`EOF`**'. La funzione restituisce il carattere fornito come argomento, dopo la conversione in maiuscolo, ammesso che ci possa essere una conversione.

File «stdlib.h»

Funzione	Descrizione

`int atoi (const char *s);`
`long int atol (const char *s);`
`long long int atoll (const char *s);`
`double atof (const char *s);`
	Converte la stringa fornita come argomento in un numero intero o in virgola mobile, come si vede dal modello sintattico.

Funzione	Descrizione

`float strtof (const char *restrict s, char **restrict p);`
`double strtod (const char *restrict s, char **restrict p);`
`long double strtold (const char *restrict s, char **restrict p);`
	Converte la stringa fornita come primo argomento in un numero in virgola mobile, come si vede dal modello sintattico. La conversione avviene fino a dove è possibile riconoscere caratteri che compongono un valore valido; se il secondo argomento è un puntatore a un puntatore valido (un puntatore a un'area di memoria che può contenere a sua volta un puntatore dal tipo '**char**'), al suo interno viene memorizzato l'indirizzo finale della scansione, a partire dal quale si trovano caratteri non decifrabili, oppure dove si trova il carattere nullo di terminazione della stringa.

Funzione	Descrizione

`long int strtol (const char *restrict s, char **restrict p, int base);`
`long long int strtoll (const char *restrict s, char **restrict p, int base);`
`unsigned long int strtoul (const char *restrict s, char **restrict p, int base);`
`unsigned long long int strtoull (const char *restrict s, char **restrict p, int base);`
	Converte la stringa fornita come primo argomento in un numero intero, come si vede dal modello sintattico, interpretando la stringa come numero espresso nella base rappresentata dal parametro ***base***. La conversione avviene fino a dove è possibile riconoscere caratteri che compongono un valore valido; se il secondo argomento è un puntatore a un puntatore valido (un puntatore a un'area di memoria che può contenere a sua volta un puntatore dal tipo '**`char`**'), al suo interno viene memorizzato l'indirizzo finale della scansione, a partire dal quale si trovano caratteri non decifrabili, oppure dove si trova il carattere nullo di terminazione della stringa.

`void srand (unsigned int seed);`	Modifica il seme per la generazione di numeri casuali attraverso la funzione ***rand()***.

Funzione	Descrizione
`int rand (void);`	Restituisce il numero casuale successivo.
`void *malloc (size_t` ***size*** `);`	Richiede l'allocazione di memoria di almeno ***size*** byte, restituendo il puntatore all'inizio della stessa se l'operazione ha successo, oppure il puntatore nullo se l'allocazione fallisce.
`void *calloc (size_t` ***nmemb*** `,` `size_t` ***size*** `);`	Richiede l'allocazione di memoria di almeno ***nmemb*** elementi da ***size*** byte ciascuno, restituendo il puntatore all'inizio della stessa se l'operazione ha successo, oppure il puntatore nullo se l'allocazione fallisce.

Funzione	Descrizione
`void *realloc (void *`***ptr***`, size_t` ***size***`);`	Richiede la riallocazione della memoria già allocata precedentemente a partire dall'indirizzo ***ptr***, in modo da avere almeno ***size*** byte, recuperando il contenuto precedente, per ciò che è possibile. La riallocazione può avvenire in corrispondenza di un indirizzo differente da quello originale, ma può fallire, restituendo così solo il puntatore nullo.
`void free (void *`***ptr***`);`	Libera la memoria allocata precedente a partire dall'indirizzo ***ptr***.
`int atexit (void (*`***func***`) (void));`	Accumula in un elenco il puntatore a una funzione che non richiede argomenti, da eseguire, assieme alle altre dell'elenco, quando viene chiamata la funzione ***exit()***. La funzione ***atexit()*** restituisce un valore numerico da intendere come *Vero* o *Falso*, a indicare se l'operazione ha avuto successo o meno.

Funzione	Descrizione
`void exit (int` ***status*** `);`	Conclude il funzionamento del programma, ma prima esegue le funzioni accumulate con l'ausilio di ***atexit()***, quindi chiude i file e infine passa il valore ricevuto come argomento in modo tale che sia restituito dal programma stesso.
`void _Exit (int` ***status*** `);`	Conclude il funzionamento del programma in modo brutale, senza occuparsi di nulla, a parte il far sì che il programma restituisca il valore indicato come argomento.
`void abort (void);`	Produce l'emissione del segnale '**`SIGABRT`**' (*abort*) che porta alla morte del processo elaborativo.
`char *getenv (const char *`***name***`);`	Restituisce il puntatore all'inizio della stringa che rappresenta il contenuto della variabile di ambiente indicata per nome, come argomento.

Funzione	Descrizione
`int system (const char *`***string***`);`	Esegue il comando indicato come argomento, attraverso il sistema operativo, restituendo il valore di uscita del comando stesso.

`void qsort (void *`***base***`,`
`size_t` ***nmemb***`,`
`size_t` ***size***`,`
`int (*`***compar***`)`
`(const void *,`
`const void *));`
	Riordina un array che inizia dall'indirizzo ***base***, essendo composto da ***size*** elementi da ***nmemb*** byte ognuno, utilizzando per il confronto la funzione ***compar***.

`void *bsearch (const void *`***key***`,`
`const void *`***base***`,`
`size_t` ***nmemb***`,`
`size_t` ***size***`,`
`int (*`***compar***`)`
`(const void *,`
`const void *));`
	Scandisce un array che inizia dall'indirizzo ***base***, essendo composto da ***size*** elementi da ***nmemb*** byte ognuno, il quale risulta già ordinato secondo la funzione ***compar***, alla ricerca della corrispondenza con il valore ****key***, la cui dimensione deve essere sempre di ***nmemb*** byte.

`int abs (int` ***j***`);`
`long int labs (long int` ***j***`);`
`long long int llabs`
`(long long int` ***j***`);`
	Restituisce il valore assoluto di ***j***.

Funzione	Descrizione
`div_t div (int` ***numeratore*****`,`** `int` ***denominatore*****`);`** `ldiv_t ldiv (long int` ***numeratore*****`,`** `long int` ***denominatore*****`);`** `lldiv_t lldiv` `(long long int` ***numeratore*****`,`** `long long int` ***denominatore*****`);`**	Restituisce il risultato della divisione dei due argomenti, in una struttura contenente il risultato intero e il resto della divisione.
`int mblen (const char *`***s***`, size_t` ***n***`);`	Restituisce la lunghezza in byte del primo carattere multibyte contenuto nella stringa fornita come primo argomento. La scansione termina comunque se raggiunge la quantità di byte indicata dal secondo argomento. Se la stringa contiene una sequenza multibyte errata o incompleta, il valore restituito è −1. Se al posto della stringa multibyte si fornisce il puntatore nullo, la funzione restituisce il valore uno o zero, a seconda che sia prevista o meno una codifica multibyte con una gestione dello stato (*shift state*).

Funzione	Descrizione
`int mbtowc (wchar_t *restrict` ***pwc*****`,`** `const char *restrict` ***s*****`,`** `size_t` ***n*****`);`**	Converte il carattere multibyte contenuto nella stringa ***s***, per un massimo di ***n*** byte, nel carattere esteso a cui punta ***pwc***, restituendo la quantità di byte utilizzati dalla stringa di origine, oppure −1 se si presentano errori. Se al posto della stringa ***s*** si mette il puntatore nullo, si ottiene un valore pari a uno o zero, a seconda che sia prevista o meno una codifica multibyte con una gestione dello stato (*shift state*).

Funzione	Descrizione
`int wctomb (char *s, wchar_t wc;`	Converte il carattere esteso ***wc*** in una sequenza multibyte che va a essere contenuta nella stringa ***s***, restituendo la quantità di byte prodotti, oppure −1 se si presentano errori. Se al posto della stringa ***s*** si mette il puntatore nullo, si ottiene un valore pari a uno o zero, a seconda che sia prevista o meno una codifica multibyte con una gestione dello stato (*shift state*).

Funzione	Descrizione
`size_t mbstowcs (wchar_t *restrict` ***pwcs*****`,`** `const char *restrict` ***s*****`,`** `size_t` ***n*****`);`**	Converte la stringa multibyte ***s*** nella stringa estesa ***pwcs***, producendo al massimo ***n*** caratteri estesi nella destinazione (incluso il carattere nullo di terminazione). La funzione restituisce la quantità di caratteri estesi copiati nella destinazione, escludendo il carattere nullo di terminazione, oppure l'equivalente di -1 in caso di errori. Se al posto della destinazione viene messo il puntatore nullo, l'operazione viene simulata ignorando il valore di ***n*** e senza memorizzare il risultato; pertanto è utile per contare lo spazio necessario nella destinazione.

Funzione	Descrizione
`size_t wcstombs (char *restrict s, wchar_t *restrict pwcs, size_t n);`	Converte la stringa estesa ***pwcs*** nella stringa multibyte ***s***, producendo al massimo ***n*** byte nella destinazione (incluso il carattere nullo di terminazione). La funzione restituisce la quantità di byte copiati nella destinazione, escludendo il carattere nullo di terminazione, oppure l'equivalente di -1 in caso di errori. Se al posto della destinazione viene messo il puntatore nullo, l'operazione viene simulata ignorando il valore di ***n*** e senza memorizzare il risultato; pertanto è utile per contare lo spazio necessario nella destinazione.

File «string.h»

«

Funzione	Descrizione
`void *memcpy (void *restrict dst, const void *restrict org, size_t n);`	Copia ***n*** caratteri a partire dall'indirizzo indicato da ***org***, per riprodurli a partire dall'indirizzo ***dst***, alla condizione che i due insiemi non risultino sovrapposti. La funzione restituisce l'indirizzo ***dst***.
`void *memmove (void *dst, const void *org, size_t n);`	Copia ***n*** caratteri a partire dall'indirizzo indicato da ***org***, per riprodurli a partire dall'indirizzo ***dst***, senza il vincolo che gli insiemi siano disgiunti. La funzione restituisce l'indirizzo ***dst***.
`char *strcpy (char *restrict dst, const char *restrict org);`	Copia la stringa ***org*** nell'array a cui punta ***dst***, includendo anche il carattere nullo di conclusione delle stringhe, alla condizione che le due stringhe non si sovrappongano. La funzione restituisce ***dst***.

Funzione	Descrizione
`char *strncpy (char *restrict dst, const char *restrict org, size_t n);`	Copia ***n*** caratteri della stringa ***org*** nell'array a cui punta ***dst***, contando tra i caratteri copiati anche il carattere nullo di conclusione delle stringhe, alla condizione che le due stringhe non si sovrappongano. Se la stringa di origine è più corta di ***n***, i caratteri mancanti sono rimpiazzati dal carattere nullo di conclusione delle stringhe. La funzione restituisce ***dst***.
`char *strcat (char *restrict dst, const char *restrict org);`	Copia la stringa ***org*** a partire dalla fine della stringa ***dst*** (sovrascrivendo il carattere nullo preesistente), alla condizione che le due stringhe non siano sovrapposte. La funzione restituisce ***dst***.

Funzione	Descrizione
`char *strncat (char *restrict dst, const char *restrict org, size_t n);`	Copia al massimo ***n*** caratteri della stringa ***org*** a partire dalla fine della stringa ***dst*** (sovrascrivendo il carattere nullo preesistente), aggiungendo alla fine il carattere nullo di terminazione, il tutto alla condizione che le due stringhe non siano sovrapposte. La funzione restituisce ***dst***.
`int memcmp (const void *s1, const void *s2, size_t n);`	Confronta i primi ***n*** caratteri delle aree di memoria a cui puntano ***s1*** e ***s2***, restituendo: un valore pari a zero se le due sequenze si equivalgono; un valore maggiore di zero se la sequenza di ***s1*** è maggiore di ***s2***; un valore minore di zero se la sequenza di ***s1*** è minore di ***s2***.
`int strcmp (const char *s1, const char *s2);`	Confronta due stringhe restituendo: un valore pari a zero se sono uguali; un valore maggiore di zero se la stringa ***s1*** è maggiore di ***s2***; un valore minore di zero se la stringa ***s1*** è minore di ***s2***.

Funzione	Descrizione
`int strcoll (const char *s1, const char *s2);`	La funzione ***strcoll()*** è analoga a ***strcmp()***, con la differenza che la comparazione avviene sulla base della configurazione locale (la categoria '**`LC_COLLATE`**'). Nel caso della configurazione locale '**`C`**' la funzione si comporta esattamente come ***strcmp()***.
`int strncmp (const char *s1, char *s2, size_t n);`	La funzione ***strncmp()*** si comporta in modo analogo a ***strcmp()***, con la differenza che la comparazione si arresta al massimo dopo ***n*** caratteri.

Funzione	Descrizione
`size_t strxfrm (char *restrict dst, const char *restrict org, size_t n);`	La funzione ***strxfrm()*** trasforma la stringa ***org*** sovrascrivendo la stringa ***dst*** in modo relativo alla configurazione locale. In pratica, la stringa trasformata che si ottiene può essere comparata con un'altra stringa trasformata nello stesso modo attraverso la funzione ***strcmp()*** ottenendo lo stesso esito che si avrebbe confrontando le stringhe originali con la funzione ***strcoll()***.
`void *memchr (const void *s, int c, size_t n);`	Cerca un carattere a partire da una certa posizione in memoria, scandendo al massimo una quantità determinata di caratteri, restituendo il puntatore al carattere trovato. Se nell'ambito specificato non trova il carattere, restituisce il puntatore nullo.

Funzione	Descrizione
`char *strchr (const char *s, int c);`	Cerca un carattere all'interno di una stringa, restituendo il puntatore al carattere trovato, oppure il puntatore nullo se la ricerca fallisce. Nella scansione viene preso in considerazione anche il carattere nullo di terminazione della stringa.
`char *strrchr (const char *s, int c);`	Cerca un carattere all'interno di una stringa, restituendo il puntatore all'ultimo carattere corrispondente trovato, oppure il puntatore nullo se la ricerca fallisce. Nella scansione viene preso in considerazione anche il carattere nullo di terminazione della stringa.
`size_t strspn (const char *s, const char *accept);`	Calcola la lunghezza massima iniziale della stringa ***s***, composta esclusivamente da caratteri contenuti nella stringa ***accept***, restituendo tale valore.

Funzione	Descrizione
`size_t strcspn (const char *s, const char *reject);`	Calcola la lunghezza massima iniziale della stringa ***s***, composta esclusivamente da caratteri differenti da quelli contenuti nella stringa ***reject***.
`char *strpbrk (const char *s, const char *accept);`	Scandisce la stringa ***s*** alla ricerca del primo carattere che risulti contenuto nella stringa ***accept***, restituendo il puntatore al carattere trovato, oppure, in mancanza di alcuna corrispondenza, il puntatore nullo.
`char *strstr (const char *string, const char *substring);`	Cerca la stringa ***substring*** nella stringa ***string*** restituendo il puntatore alla prima corrispondenza trovata (nella stringa ***string***). Se la corrispondenza non c'è, la funzione restituisce il puntatore nullo.

Funzione	Descrizione
`char *strtok (char *restrict string, const char *restrict delim);`	Serve a suddividere una stringa in unità, definite *token*, specificando un elenco di caratteri da intendere come delimitatori, in una seconda stringa. La funzione va usata in fasi successive, fornendo solo inizialmente la stringa da suddividere che continua poi a essere utilizzata se al suo posto viene fornito il puntatore nullo. La funzione restituisce, di volta in volta, il puntatore alla sottostringa contenente l'unità individuata, oppure il puntatore nullo, se non può trovarla.
`void *memset (void *s, int c, size_t n);`	Inizializza una certa area di memoria, a partire dall'indirizzo ***s***, con la ripetizione del carattere ***c***, tradotto in un carattere senza segno, copiandolo per ***n*** volte. La funzione restituisce ***s***.
`char *strerror (int errnum);`	Trasforma un numero nella descrizione del tipo di errore corrispondente.

Funzione	Descrizione
`size_t strlen (const char *s);`	Calcola la lunghezza di una stringa, escludendo dal conteggio il carattere nullo di terminazione.

File «time.h»

Funzione	Descrizione
`clock_t clock (void);`	Restituisce il tempo di CPU espresso in unità '**`clock_t`**', utilizzato dal processo elaborativo a partire dall'avvio del programma. Se la funzione non è in grado di dare questa indicazione, allora restituisce il valore −1, o più precisamente '**`(clock_t) (-1)`**'.
`time_t time (time_t *timer);`	Determina il tempo attuale secondo il calendario del sistema operativo, restituendolo nella forma del tipo '**`time_t`**'. Se il puntatore di tipo '**`time_t *`**' è valido, la stessa informazione che viene restituita viene anche memorizzata nell'indirizzo indicato da tale puntatore.

Funzione	Descrizione
`double difftime (time_t` ***time1*****`,`** `time_t` ***time0***`);`	Calcola la differenza tra due date, espresse in forma '**`time_t`**', restituendo l'intervallo in secondi.

Funzione	Descrizione
`time_t mktime (struct tm *`***timeptr***`);`	Riceve come argomento il puntatore a una variabile strutturata di tipo '**`struct tm`**', contenente le informazioni sull'ora locale, e determina il valore di quella data secondo la rappresentazione interna, di tipo '**`time_t`**'. La funzione non tiene conto del giorno della settimana e del giorno dell'anno; inoltre, ammette anche valori al di fuori degli intervalli stabiliti per i vari membri della struttura; infine, considera un valore negativo per il membro '***`timeptr`***`->tm_isdst`' come la richiesta di determinare se sia o meno in vigore l'ora estiva per la data indicata. Se la funzione non è in grado di restituire un valore rappresentabile nel tipo '**`time_t`**', o comunque se non può eseguire il suo compito, restituisce il valore −1, o più precisamente '**`(time_t) (-1)`**'. Se invece tutto procede regolarmente, la funzione provvede anche a correggere i valori dei vari membri della struttura e a ricalcolare il giorno della settimana e dell'anno.

Funzione	Descrizione
`struct tm *gmtime (const time_t *timer);`	Converte una data espressa nella forma del tipo '**`time_t`**', in una data suddivisa nella struttura '**`tm`**', relativa al tempo universale coordinato (UTC).
`struct tm *localtime (const time_t *timer);`	Converte una data espressa nella forma del tipo '**`time_t`**', in una data suddivisa nella struttura '**`tm`**', relativa all'ora locale.
`char *asctime (const struct tm *timeptr);`	Converte un'informazione data-orario, espressa nella forma di una struttura '**`struct tm`**', in una stringa che esprime l'ora locale, usando però una rappresentazione fissa in lingua inglese.
`char *ctime (const time_t *timer);`	Converte un'informazione data-orario, espressa nella forma del tipo '**`time_t`**', in una stringa che esprime l'ora locale, usando però una rappresentazione fissa in lingua inglese.

Funzione	Descrizione
`size_t strftime (char *restrict` ***s***`, size_t` ***maxsize***`, const char *restrict` ***format***`, const struct tm *restrict` ***timeptr***`);`	Interpreta il contenuto di una struttura di tipo '**`struct tm`**' e lo traduce in un testo, secondo una stringa di composizione libera. Il comportamento è affine a quello di ***printf()***, dove l'input è costituito dalla struttura contenente le informazioni data-orario.

Specificatore di conversione	Corrispondenza
`%C`	*century* Il secolo, ottenuto dividendo l'anno per 100 e ignorando i decimali.
`%y` `%Y`	*year* L'anno: nel primo caso si mostrano solo le ultime due cifre, mentre nel secondo si mostrano tutte.
`%b` | `%h` `%B`	Rispettivamente, il nome abbreviato e il nome per esteso del mese.
`%m`	*month* Il numero del mese, da 01 a 12.

Specificatore di conversione	Corrispondenza
`%d` `%e`	*day* Il giorno del mese, in forma numerica, da 1 a 31, utilizzando sempre due cifre: nel primo caso si aggiunge eventualmente uno zero; nel secondo si aggiunge eventualmente uno spazio.
`%a` `%A`	Rispettivamente, il nome abbreviato e il nome per esteso del giorno della settimana.
`%H` `%L`	*hour* L'ora, espressa rispettivamente a 24 ore e a 12 ore.
`%p`	La sigla da usare, secondo la configurazione locale, per specificare che si tratta di un'ora antimeridiana o pomeridiana. Nella convenzione inglese si ottengono, per esempio, le sigle «AM» e «PM».
`%r`	L'ora espressa a 12 ore, completa dell'indicazione se trattasi di ora antimeridiana o pomeridiana, secondo le convenzioni locali.
`%R`	L'ora e i minuti, equivalente a '**`%H:%M`**'.
`%M`	*minute* I minuti, da 00 a 59.
`%S`	*second* I secondi, espresso con valori da 00 a 60.
`%T`	*time* L'ora, i minuti e i secondi, equivalente a '**`%H:%M:%S`**'.

Specificatore di conversione	Corrispondenza
`%z` `%Z`	*time zone* La rappresentazione del fuso orario, nel primo caso come distanza dal tempo coordinato universale (UTC), mentre nel secondo si usa una rappresentazione conforme alla configurazione locale.
`%j`	*julian* Il giorno dell'anno, usando sempre tre cifre numeriche: da 001 a 366.
`%g` `%G`	L'anno a cui appartiene la settimana secondo lo standard ISO 8601: nel primo caso si mostrano solo le ultime due cifre, mentre nel secondo si ha l'anno per esteso. Secondo lo standard ISO 8601 la settimana inizia con lunedì e la prima settimana dell'anno è quella che include il 4 gennaio.
`%V`	Il numero della settimana secondo lo standard ISO 8601. I valori vanno da 01 a 53. Secondo lo standard ISO 8601 la settimana inizia con lunedì e la prima settimana dell'anno è quella che include il 4 gennaio.
`%u` `%w`	Il giorno della settimana, espresso in forma numerica, dove, rispettivamente, si conta da 1 a 7, oppure da 0 a 6. Zero e sette rappresentano la domenica; uno è il lunedì.
`%U` `%W`	Il numero della settimana, contando, rispettivamente, dalla prima domenica o dal primo lunedì di gennaio. Si ottengono cifre da 00 a 53.
`%x`	La data, rappresentata secondo le convenzioni locali.
`%X`	L'ora, rappresentata secondo le convenzioni locali.

Specificatore di conversione	Corrispondenza
`%c`	La data e l'ora, rappresentate secondo le convenzioni locali.
`%D`	*date* La data, rappresentata come '**`%m/%d/%Y`**'.
`%F`	La data, rappresentata come '**`%Y-%m-%d`**'.
`%n`	Viene rimpiazzato dal codice di interruzione di riga.
`%t`	Viene rimpiazzato da una tabulazione orizzontale.
`%%`	Viene rimpiazzato da un carattere di percentuale.

File «stdio.h» per la gestione dei file e degli errori

«

Macro-variabile	Significato mnemonico	Descrizione
`_IOFBF`	*input output fully buffered*	Indica simbolicamente la richiesta di utilizzo di una memoria tampone a blocchi.
`_IOLBF`	*input output line buffered*	Indica simbolicamente la richiesta di utilizzo di una memoria tampone gestita a righe di testo.
`_IONBF`	*input output with no buffering*	Indica simbolicamente la richiesta di non utilizzare alcuna memoria tampone.

Macro-variabile	Significato mnemonico	Descrizione
`BUFSIZE`	*buffer size*	Rappresenta la dimensione predefinita della memoria tampone.
`EOF`	*end of file*	È un numero intero di tipo '**`int`**', negativo, che rappresenta il raggiungimento della fine del file. È in pratica ciò che si ottiene leggendo oltre la fine del file.
`FOPEN_MAX`	*file open max*	Il numero di file che un processo elaborativo può aprire simultaneamente, in base alle limitazioni poste dal sistema operativo.
`FILENAME_MAX`		La dimensione di un array di elementi '**`char`**', tale da essere abbastanza grande da contenere il nome del file più lungo (incluse le eventuali sequenze multibyte) che il sistema consenta di gestire.
`L_tmpnam`	*temporary name*	La dimensione di un array di elementi '**`char`**', tale da essere abbastanza grande da contenere il nome di un file temporaneo generato dalla funzione ***tmpnam()***.
`SEEK_CUR`	*seek current*	Indica di eseguire un posizionamento a partire dalla posizione corrente del file.

Macro-variabile	Significato mnemonico	Descrizione
`SEEK_END`		Indica di eseguire un posizionamento a partire dalla fine di un file.
`SEEK_SET`		Indica di eseguire un posizionamento a partire dall'inizio di un file.
`TMP_MAX`		Rappresenta la quantità massima di nomi di file differenti che possono essere generati dalla funzione ***tmpnam()***.

Modalità di accesso ai file	Mnemonico	Descrizione
`r`	*read*	Accesso in sola lettura di un file di testo.
`w`	*write*	Accesso a un file di testo in scrittura, che implica la creazione del file all'apertura, ovvero il suo troncamento a zero, se esiste già.
`a`	*append*	Accesso a un file di testo in aggiunta, che implica la creazione del file all'apertura, ovvero la sua estensione se esiste già.

`rb`
`wb`
`ab`
	binary	Accesso in lettura, scrittura o aggiunta, ma di tipo binario.

Modalità di accesso ai file	Mnemonico	Descrizione

`r+`
`w+`
`a+`
	update	Accesso a un file di testo in lettura, scrittura o aggiunta, assieme alla modalità di aggiornamento. In pratica, con la lettura è consentita anche la scrittura; con la scrittura e l'aggiunta è consentita anche la rilettura.

`rb+` | `r+b`
`wb+` | `w+b`
`ab+` | `a+b`
		Accesso a un file binario in lettura, scrittura o aggiunta, assieme alla modalità di aggiornamento. In pratica, con la lettura è consentita anche la scrittura; con la scrittura e l'aggiunta è consentita anche la rilettura. Si può osservare che il segno '**+**' può essere messo indifferentemente in mezzo o alla fine.

Funzione	Descrizione
`int remove (const char *`***filename***`);`	Cancella il file il cui nome viene fornito come argomento. Il nome del file va espresso secondo le convenzioni del sistema operativo. Restituisce zero se l'operazione ha successo, altrimenti un valore differente.

Funzione	Descrizione
`int rename (const char *old, const char *new);`	Cambia il nome del file indicato come primo argomento, in modo che assuma quello del secondo argomento. Se l'operazione avviene con successo restituisce zero, altrimenti produce un valore differente.
`FILE *tmpfile (void);`	Crea e apre un file temporaneo binario in aggiornamento ('**wb+**'), restituendone il puntatore. Se il file temporaneo non può essere creato, la funzione restituisce il puntatore nullo.
`char *tmpnam (char *s);`	Genera il nome di un file che può essere usato come file temporaneo. La funzione richiede come argomento un array di almeno '**L_tmpnam**' caratteri, da usare per scriverci il nome e per restituirne il puntatore. Se alla funzione viene passato il puntatore nullo, allora questa usa un'area di memoria statica che viene sovrascritta a ogni chiamata successiva della funzione stessa. Se la funzione non può eseguire il suo lavoro, restituisce il puntatore nullo.

Funzione	Descrizione
`FILE *fopen (const char *restrict` ***filename*****,** `const char *restrict` ***io_mode*****);**	Apre il file indicato come primo argomento, secondo la modalità espressa dalla stringa che costituisce il secondo argomento, restituendo il puntatore che ne rappresenta il flusso aperto. Se l'operazione fallisce la funzione restituisce il puntatore nullo e aggiorna il valore della variabile globale ***errno***.
`FILE *freopen (const char *restrict` ***filename*****,** `const char *restrict` ***io_mode*****,** `FILE *restrict` ***stream*****);**	Apre il file indicato come primo argomento, secondo la modalità espressa dalla stringa che costituisce il secondo argomento, utilizzando il flusso di file individuato dal puntatore che costituisce l'ultimo argomento, restituendo lo stesso puntatore. Se il puntatore indicato come ultimo argomento riguarda un flusso di file ancora aperto, questo viene chiuso e quindi riaperto. Se l'operazione fallisce la funzione restituisce il puntatore nullo e aggiorna il valore della variabile globale ***errno***. Lo scopo di questa funzione è quello di ridirigere i flussi di file, associando file differenti.

Funzione	Descrizione
`int fclose (FILE *`***stream***`);`	Chiude il flusso di file individuato dal puntatore che costituisce l'argomento della funzione. La funzione restituisce il valore zero se l'operazione ha successo, altrimenti produce il valore corrispondente alla macrovariabile ***EOF***. Va sottolineato che un flusso già chiuso non deve essere chiuso nuovamente, perché in tal caso l'effetto che se ne produce è imprecisato.

Funzione	Descrizione
`int setvbuf (FILE *restrict` ***stream*****,** `char *restrict` ***buffer*****,** `int` ***buf_mode*****,** `size_t` ***size*****);**	Attribuisce una memoria tampone a un file che è appena stato aperto e per il quale non è ancora stato eseguito alcun accesso. Il primo argomento della funzione è il puntatore al flusso relativo e il secondo è il puntatore all'inizio dell'array di caratteri da usare come memoria tampone. Se al posto del riferimento alla memoria tampone si indica un puntatore nullo, si intende che la funzione debba allocare automaticamente lo spazio necessario; se invece l'array viene fornito, è evidente che deve rimanere disponibile per tutto il tempo in cui il flusso rimane aperto. Il terzo argomento atteso dalla funzione è un numero che esprime la modalità di funzionamento della memoria tampone. Questo numero viene fornito attraverso l'indicazione di una tra le macro-variabili ***_IOFBF***, ***_IOLBF*** e ***_IONBF***. Il quarto argomento indica la dimensione dell'array da usare come memoria tampone: se l'array viene fornito effettivamente, si tratta della dimensione che può essere utilizzata; altrimenti è la dimensione richiesta per l'allocazione automatica. La funzione restituisce zero se l'operazione richiesta è eseguita con successo; diversamente restituisce un valore differente.

Funzione	Descrizione
`void setbuf (FILE *restrict` ***stream*****`,`** `char *restrict` ***buffer*****`);`	Si tratta di una versione semplificata di '**`setvbuf()`**' che non restituisce alcun valore, che prevede implicitamente una modalità di gestione completa della memoria tampone ('**`_IOFBF`**'), che richiede implicitamente un array di '**`BUFSIZ`**' elementi. Anche in questo caso, se l'argomento corrispondente al parametro ***buffer*** è un puntatore nullo, l'allocazione avviene in modo automatico.
`int fflush (FILE *`***stream***`);`	Scarica la memoria tampone del flusso di file indicato, procedendo così alla memorizzazione dei dati rimasti in sospeso. Restituisce zero se l'operazione viene completata con successo, altrimenti restituisce il valore corrispondente alla macro-variabile ***EOF***.

`int fgetc (FILE *`***stream***`);`
 	
`int getc (FILE *`***stream***`);`
	Legge un carattere (senza segno) dal flusso di file indicato come argomento e ne restituisce il valore numerico (positivo). Se l'operazione fallisce, la funzione restituisce il valore corrispondente alla macro-variabile ***EOF***. Tradizionalmente, ***fgetc()*** è sempre una funzione, mentre ***getc()*** potrebbe essere una macroistruzione che valuta anche più volte l'espressione che costituisce l'argomento.

Funzione	Descrizione
`int ungetc (int c, FILE *stream);`	Rimanda indietro il carattere ***c*** nel flusso di file ***stream***; in altri termini dovrebbe annullare l'effetto dell'ultima chiamata a una funzione ***fgetc()*** o ***getc()***.

`int fputc (int c, FILE *stream);`
 	
`int putc (int c, FILE *stream);`
	Scrive un carattere, rappresentato dal primo argomento, nel flusso di file indicato come secondo argomento, restituendo lo stesso valore del carattere scritto, se l'operazione si conclude con successo, oppure il valore corrispondente a '**EOF**' se l'operazione fallisce. Tradizionalmente, ***fputc()*** è sempre una funzione, mentre ***putc()*** potrebbe essere una macroistruzione che valuta anche più volte le espressioni che costituiscono gli argomenti.

`int putchar (int c);`	Scrive un carattere, rappresentato dall'argomento, nello standard output, restituendo lo stesso valore del carattere scritto, se l'operazione si conclude con successo, oppure il valore corrispondente a '**EOF**' se l'operazione fallisce. Tradizionalmente si tratta di una macroistruzione che valuta anche più volte l'espressione che costituisce l'argomento.

Funzione	Descrizione

`char *fgets (char *restrict s,`
`int n,`
`FILE *restrict stream);`
	Legge al massimo $n-1$ caratteri (elementi '**`char`**') attraverso il flusso di file ***stream***, copiandoli in memoria a partire dall'indirizzo ***s*** e aggiungendo alla fine il carattere nullo di terminazione delle stringhe. La lettura si esaurisce prima di $n-1$ caratteri se viene incontrato il codice di interruzione di riga, il quale viene rappresentato nella stringa a cui punta ***s***, ovvero se si raggiunge la fine del file. In ogni caso, la stringa ***s*** viene terminata correttamente con il carattere nullo.
	La funzione restituisce la stringa ***s*** se la lettura avviene con successo, ovvero se ha prodotto almeno un carattere; altrimenti, il contenuto dell'array a cui punta ***s*** non viene modificato e la funzione restituisce il puntatore nullo. Se si creano errori imprevisti, la funzione potrebbe restituire il puntatore nullo, ma senza garantire che l'array ***s*** sia rimasto intatto.

Funzione	Descrizione
`int fputs (const char *restrict s, FILE *restrict stream);`	Copia la stringa a cui punta ***s*** nel file rappresentato dal flusso di file ***stream***. La copia della stringa avviene escludendo però il carattere nullo di terminazione. Va osservato che questa funzione, pur essendo contrapposta evidentemente a '**`fgets()`**', **non conclude la riga** del file, ovvero, non aggiunge il codice di interruzione di riga. Per ottenere la conclusione della riga di un file di testo, occorre inserire nella stringa, espressamente, il carattere '**`\n`**'. La funzione restituisce il valore rappresentato da '**`EOF`**' se l'operazione di scrittura produce un errore; altrimenti restituisce un valore positivo qualunque.
`char *gets (char *s);`	Legge una riga dallo standard input, copiandola in memoria a partire dall'indirizzo ***s*** e aggiungendo alla fine il carattere nullo di terminazione delle stringhe. Per il resto, il funzionamento è conforme a quello di ***fgets()***.
`int puts (const char *s);`	Copia la stringa a cui punta ***s*** nello standard output, aggiungendo in coda il codice di interruzione di riga. Per il resto, il funzionamento è analogo a quello di ***fputs()***.

Funzione	Descrizione
`size_t fread (void *restrict` ***ptr*****`,`** `size_t` ***size*****`,`** `size_t` ***nmemb*****`,`** `FILE *restrict` ***stream*****`);`**	Legge dal flusso di file ***stream***, ***nmemb*** blocchi da ***size*** byte, copiando questi dati in memoria a partire dall'indirizzo ***ptr***. Restituisce la quantità di blocchi da '**`size`**' byte che sono stati copiati con successo; pertanto, se questo valore è inferiore a ***nmemb***, si è verificato un problema.
`size_t fwrite` `(const void *restrict` ***ptr*****`,`** `size_t` ***size*****`,`** `size_t` ***nmemb*****`,`** `FILE *restrict` ***stream*****`);`**	Scrive nel flusso di file ***stream***, ***nmemb*** blocchi da ***size*** byte, leggendo questi dati dalla memoria a partire dall'indirizzo ***ptr***. Restituisce la quantità di blocchi da '**`size`**' byte che sono stati copiati con successo; pertanto, se questo valore è inferiore a ***nmemb***, si è verificato un problema.

Funzione	Descrizione

`int fseek (FILE *`***stream***`,`
`long int` ***offset***`,`
`int` ***whence***`);`
	Sposta la posizione corrente relativa al flusso di file ***stream*** (associato preferibilmente a un file binario), nella nuova posizione determinata dai parametri ***whence*** e ***offset***. Il parametro ***whence*** viene fornito attraverso una macro-variabile che può essere ***SEEK_SET***, ***SEEK_CUR*** o ***SEEK_END***, indicando rispettivamente l'inizio del file, la posizione corrente o la fine del file. Dalla posizione indicata dal parametro ***whence*** viene aggiunta, algebricamente, la quantità di byte indicata dal parametro ***offset***. La funzione restituisce zero se può eseguire l'operazione, altrimenti dà un risultato diverso.

`long int ftell (FILE *`***stream***`);`	Restituisce la posizione corrente del flusso di file indicato come argomento. Questo valore può essere usato con ***fseek()***, al posto dello scostamento (il parametro ***offset***), indicando come posizione di riferimento l'inizio del file, ovvero '**`SEEK_SET`**'. Se la funzione non riesce a fornire la posizione, restituisce il valore −1 (tradotto in '**`long int`**') e annota il fatto nella variabile ***errno***.

Funzione	Descrizione
`void rewind (FILE *`***stream***`);`	Riposiziona il flusso di file all'inizio. In pratica è come utilizzare la funzione ***fseek()*** specificando uno scostamento pari a zero a partire da '**`SEEK_SET`**', ignorando il valore restituito.
`int fgetpos (FILE *restrict` ***stream***`,` `fpos_t *restrict` ***pos***`);`	Memorizza nella variabile a cui punta il parametro ***pos*** le informazioni sulla posizione corrente del file, assieme allo stato di interpretazione relativo alle sequenze multibyte. Restituisce zero se l'operazione è stata compiuta con successo, altrimenti dà un altro valore
`int fsetpos (FILE *`***stream***`,` `const fpos_t *`***pos***`);`	Utilizza la variabile a cui punta ***pos*** per ripristinare la posizione memorizzata, assieme allo stato di avanzamento dell'interpretazione di una sequenza multibyte. Restituisce zero se l'operazione è stata compiuta con successo, altrimenti dà un altro valore e aggiorna la variabile ***errno***.
`void clearerr (FILE *`***stream***`);`	Azzera gli indicatori di errore e di fine file per il flusso di file indicato come argomento, senza restituire alcunché.
`int feof (FILE *`***stream***`);`	Controlla lo stato dell'indicatore di fine file per il flusso di file indicato. Se questo non è attivo restituisce zero, altrimenti restituisce un valore diverso da zero.

Funzione	Descrizione
`int ferror (FILE *`***stream***`);`	Controlla lo stato dell'indicatore di errore per il flusso di file indicato. Se questo non è attivo restituisce zero, altrimenti restituisce un valore diverso da zero.
`void perror (const char *`***s***`);`	Prende in considerazione la variabile ***errno*** e cerca di tradurla in un messaggio testuale da emettere attraverso lo standard error (con tanto di terminazione della riga, in modo da riposizionare a capo il cursore). Se il parametro ***s*** corrisponde a una stringa non vuota, il testo di questa viene posto anteriormente al messaggio, separandolo con due punti e uno spazio (‘: ’). Il contenuto del messaggio è lo stesso che si otterrebbe con la funzione ***strerror()***, fornendo come argomento la variabile ***errno***.

File «stdio.h» per la composizione dell'output

«

`%` [***simbolo***] [***n_ampiezza***] [***. n_precisione***] [`hh` | `h` | `l` | `ll` | `j` | `z` | `t` | `L`] ***tipo***

Simbolo	Tipo di argomento	Conversione applicata
`%…d` `%…i`	`int`	Numero intero con segno da rappresentare in base dieci.

Simbolo	Tipo di argomento	Conversione applicata
`%…u`	`unsigned int`	Numero intero senza segno da rappresentare in base dieci.
`%…o`	`unsigned int`	Numero intero senza segno da rappresentare in ottale (senza lo zero iniziale che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
`%…x` `%…X`	`unsigned int`	Numero intero senza segno da rappresentare in esadecimale (senza il prefisso '**`0x`**' o '**`0X`**' che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
`%…c`	`int`	Un carattere singolo, dopo la conversione in '**`unsigned char`**'.
`%…s`	`char *`	Una stringa.
`%…f`	`double`	Un numero a virgola mobile, da rappresentare in notazione decimale fissa: ***[-] iii . dddddd***
`%…e` `%…E`	`double`	Un numero a virgola mobile, da rappresentare in notazione esponenziale: ***[-] i . dddddd*`e`*$\pm$***xx*** ***[-] i . dddddd*`E`*$\pm$***xx***

Simbolo	Tipo di argomento	Conversione applicata
`%…g` `%…G`	`double`	Un numero a virgola mobile, rappresentato in notazione decimale fissa o in notazione esponenziale, a seconda di quale si presti meglio in base ai vincoli posti da altri componenti dello specificatore di conversione.
`%p`	`void *`	Un puntatore generico rappresentato in qualche modo in forma grafica.
`%n`	`int *`	Questo specificatore non esegue alcuna conversione e si limita a memorizzare un valore intero (di tipo '**`int`**') nella variabile a cui punta l'argomento. Per la precisione, viene memorizzata la quantità di caratteri generati fino a quel punto dalla conversione.
`%%`		Questo specificatore si limita a produrre un carattere di percentuale ('**`%`**') che altrimenti non sarebbe rappresentabile.

Simbolo	Corrispondenza

`%+…`
`%#+…`
`%+0`***ampiezza***`…`
`%#+0`***ampiezza***`…`
	Il segno «+» fa sì che i numeri con segno lo mostrino anche se è positivo. Può combinarsi con lo zero e il cancelletto.

Simbolo	Corrispondenza
%0***ampiezza***... %+0***ampiezza***... %#0***ampiezza***... %#+0***ampiezza***...	Lo zero fa sì che siano inseriti degli zeri a sinistra per allineare a destra il valore, nell'ambito dell'ampiezza specificata. Può combinarsi con il segno «+» e il cancelletto.
%***ampiezza***... % ***ampiezza***...	In mancanza di uno zero iniziale, in presenza dell'indicazione dell'ampiezza, il valore viene allineato a destra usando degli spazi. È possibile esprimere esplicitamente l'intenzione di usare gli spazi mettendo proprio uno spazio, ma in generale non è richiesto. Se si mette lo spazio letteralmente, questo non è poi compatibile con lo zero, mentre le combinazioni con gli altri simboli sono ammissibili.
%−***ampiezza***... %−+***ampiezza***... %#−***ampiezza***... %#−+***ampiezza***...	Il segno meno, usato quando la conversione prevede l'uso di una quantità fissa di caratteri con un valore che appare di norma allineato a destra, fa sì che il risultato sia allineato a sinistra. Il segno meno si può combinare il segno «+» e il cancelletto.

Simbolo	Corrispondenza
`%#…`	Il cancelletto richiede una modalità di rappresentazione alternativa, ammesso che questa sia prevista per il tipo di conversione specificato. È compatibili con gli altri simboli, ammesso che il suo utilizzo serva effettivamente per ottenere una rappresentazione alternativa.

Simbolo	Tipo	Simbolo	Tipo

`%…hhd`
`%…hhi`
	`signed char`	`%…hhu`
		`%…hho`
		`%…hhx` | `%…hhX`
			`unsigned char`

`%…hd`
`%…hi`
	`short int`	`%…hu`
		`%…ho`
		`%…hx` | `%…hX`
			`unsigned short int`

`%…ld`
`%…li`
	`long int`	`%…lu`
		`%…lo`
		`%…lx` | `%…lX`
			`unsigned long int`

`%…lc`	`wint_t`	`%…ls`	`wchar_t *`

Simbolo	Tipo	Simbolo	Tipo
%…lld %…lli	long long int	%…llu %…llo %…llx| %…llX	unsigned long long int
%…jd %…ji	intmax_t	%…ju %…jo %…jx|%…jX	uintmax_t
%…zd %…zi	size_t	%…zu %…zo %…zx|%…zX	size_t
%…td %…ti	ptrdiff_t	%…tu %…to %…tx|%…tX	ptrdiff_t
%…Le|%…LE %…Lf|%…LF %…Lg|%…LG	long double		

Funzione	Descrizione
`int sprintf (char *restrict s, const char *restrict format, ...);`	Produce il risultato della composizione memorizzandolo a partire dal puntatore costituito dal primo parametro (*s*) e aggiungendo il carattere nullo di terminazione.
`int snprintf (char *restrict s, size_t n, const char *restrict format, ...);`	Produce al massimo $n-1$ caratteri, aggiungendo sempre il carattere nullo di terminazione.
`int fprintf (FILE *restrict stream, const char *restrict format, ...);`	Scrive il risultato della composizione attraverso il flusso di file *stream*.
`int printf (const char *restrict format, ...);`	Scrive il risultato della composizione attraverso lo standard output.
`int vsprintf (char *restrict s, const char *restrict format, va_list arg);`	Come la funzione *sprintf()*, ricevendo gli argomenti variabili attraverso un puntatore al loro inizio.

Funzione	Descrizione
`int vsnprintf (char *restrict` ***s***`, size_t` ***n***`, const char *restrict` ***format***`, va_list` ***arg***`);`	Come la funzione ***snprintf()***, ricevendo gli argomenti variabili attraverso un puntatore al loro inizio.
`int vfprintf (FILE *restrict` ***stream***`, const char *restrict` ***format***`, va_list` ***arg***`);`	Come la funzione ***fprintf()***, ricevendo gli argomenti variabili attraverso un puntatore al loro inizio.
`int vprintf (const char *restrict` ***format***`, va_list` ***arg***`);`	Come la funzione ***printf()***, ricevendo gli argomenti variabili attraverso un puntatore al loro inizio.

File «stdio.h» per l'interpretazione dell'input

«

`%[*] [`***n_ampiezza***`] [hh|h|l|ll|j|z|t|L]` ***tipo***

Simbolo	Tipo di argomento	Conversione applicata
`%…d`	`int *`	Numero intero con segno rappresentato in base dieci.
`%…i`	`int *`	Numero intero con segno rappresentare in base dieci o in base otto, avendo come prefisso uno zero, oppure in base sedici, avendo come prefisso '**`0x`**' o '**`0X`**'.
`%…u`	`unsigned int *`	Numero intero senza segno rappresentato in base dieci.

Simbolo	Tipo di argomento	Conversione applicata
`%…o`	`unsigned int *`	Numero intero senza segno rappresentato in ottale (con o senza lo zero iniziale).
`%…x`	`unsigned int *`	Numero intero senza segno rappresentato in esadecimale (con o senza il prefisso '**`0x`**' o '**`0X`**').
`%…c`	`char *`	Interpreta un solo carattere, o più caratteri se si specifica l'ampiezza. Nella lettura contano anche gli spazi o qualunque altro carattere e non viene aggiunto il carattere nullo di terminazione.
`%…s`	`char *`	Interpreta una sequenza di caratteri che non siano spazi, aggiungendo alla fine il carattere nullo di terminazione.

`%…a`
`%…e`
`%…f`
`%…g`
	`double *`	Un numero a virgola mobile rappresentato in notazione decimale fissa o in notazione esponenziale:
		[`-`**]** ***iii . dddddd***
		[`-`**]** ***i . dddddd*****`e`**$\pm$***xx***
		[`-`**]** ***i . dddddd*****`E`**$\pm$***xx***

`%p`	`void *`	Interpreta il valore di un puntatore che sia rappresentato nello stesso modo in cui farebbe la funzione '**`printf("%p",`** ***puntatore*****`)`**'.

Simbolo	Tipo di argomento	Conversione applicata
`%n`	`int *`	Questo specificatore non esegue alcuna conversione e si limita a memorizzare la quantità di caratteri (‘**`char`**’) letti fino a quel punto.
`%…[…]`	`char *`	Interpreta una stringa non vuota contenente solo i caratteri elencati tra parentesi quadre, aggiungendo alla fine il carattere nullo di terminazione. Se tra i caratteri si cerca anche la parentesi quadra chiusa, questa va messa all’inizio dell’elenco: ‘**`%…[]…]`**’.
`%…[^…]`	`char *`	Interpreta una stringa non vuota contenente solo caratteri diversi da quelli elencati tra parentesi quadre, aggiungendo alla fine il carattere nullo di terminazione. Se tra i caratteri da escludere si vuole indicare anche la parentesi quadra chiusa, questa va messa all’inizio dell’elenco: ‘**`%…[^]…]`**’.
`%%`		Interpreta un carattere di percentuale tra i dati in ingresso, ma senza memorizzare alcunché.

Simbolo	Tipo	Simbolo	Tipo
%…hhd %…hhi	signed char *	%…hhu %…hho %…hhx %…hhn	unsigned char *
%…hd %…hi	short int *	%…hu %…ho %…hx %…hn	unsigned short int *
%…ld %…li	long int *	%…lu %…lo %…lx %…ln	unsigned long int *
		%…lc %…ls %…lc %…l[…]	wchar_t *

Simbolo	Tipo	Simbolo	Tipo
%…lld %…lli	long long int *	%…llu %…llo %…llx %…lln	unsigned long long int *
%…jd %…ji	intmax_t *	%…ju %…jo %…jx %…jn	uintmax_t *
%…zd %…zi	size_t *	%…zu %…zo %…zx %…zn	size_t *
%…td %…ti	ptrdiff_t *	%…tu %…to %…tx %…tn	ptrdiff_t *

Simbolo	Tipo	Simbolo	Tipo

`%…Le`
`%…Lf`
`%…Lg`
	`long double *`		

Funzione	Descrizione

`int fscanf`
`(FILE *restrict` ***stream*****,**
`const char *restrict` ***format*****,**
`...);`
	Scandisce l'input proveniente dal flusso_di_file che costituisce il primo parametro (***stream***), restituendo la quantità di valori assegnati alle variabili rispettive, oppure il valore corrispondente alla macro-variabile ***EOF*** nel caso si verifichi un errore prima di qualunque conversione.

`int sscanf`
`(const char *restrict` ***s*****,**
`const char *restrict` ***format*****,**
`...);`
	Scandisce il contenuto della stringa indicata come primo parametro (***s***), restituendo la quantità di valori assegnati alle variabili rispettive, oppure il valore corrispondente alla macro-variabile ***EOF*** nel caso si verifichi un errore prima di qualunque conversione.

Funzione	Descrizione
`int scanf (const char *restrict` ***format*****`,`** `...);`	Scandisce lo standard input, restituendo la quantità di valori assegnati alle variabili rispettive, oppure il valore corrispondente alla macro-variabile ***EOF*** nel caso si verifichi un errore prima di qualunque conversione.
`int vfscanf (FILE *restrict` ***stream*****`,`** `const char *restrict` ***format*****`,`** `va_list` ***arg*****`);`**	Come ***fscanf()***, con la differenza che gli argomenti variabili sono sostituiti da un puntatore al loro inizio.
`int vsscanf (const char *restrict` ***s*****`,`** `const char *restrict` ***format*****`,`** `va_list` ***arg*****`);`**	Come ***sscanf()***, con la differenza che gli argomenti variabili sono sostituiti da un puntatore al loro inizio.
`int vscanf (const char *restrict` ***format*****`,`** `va_list` ***arg*****`);`**	Come ***scanf()***, con la differenza che gli argomenti variabili sono sostituiti da un puntatore al loro inizio.

File «assert.h»

Macroistruzione	Descrizione
`void assert (`***tipo_scalare espressione***`);`	Nell'uso di questa macroistruzione, in pratica si scrive solo l'espressione tra parentesi, senza indicare espressamente il tipo scalare. Se l'espressione si traduce in un valore pari a zero, l'asserzione fallisce e viene mostrato un messaggio di errore, con le informazioni necessarie per risalire alla posizione nel file sorgente.

File «stddef.h»

Macroistruzione	Descrizione
`size_t offsetof (`***type, member***`);`	Restituisce lo scostamento che separa un membro di una struttura dall'inizio della stessa.

«

Funzione	Descrizione
`char *setlocale (int` ***category***`, NULL);`	Restituisce un puntatore alla stringa che descrive la configurazione locale corrente, riferita alla categoria specificata. Se non è in grado di fornire l'informazione, fornisce un puntatore nullo.
`char *setlocale (int` ***category***`,` `const char *`***locale***`);`	Imposta la configurazione locale, secondo il contenuto della stringa che costituisce il secondo parametro. La funzione restituisce un puntatore che descrive la stessa configurazione, oppure, se l'operazione fallisce, restituisce il puntatore nullo.

Macro-variabile	Descrizione
`LC_ALL`	Individua simultaneamente tutte le categorie relative alla configurazione locale.
`LC_COLLATE`	Categoria che definisce l'ordine alfabetico dei caratteri tipografici.
`LC_CTYPE`	Categoria che definisce il modo di raggruppare i caratteri tipografici per tipologia.
`LC_MONETARY`	Categoria che definisce le convenzioni legate alla rappresentazione dei valori che esprimono importi in valuta.

Macro-variabile	Descrizione
`LC_NUMERIC`	Categoria che definisce il modo in cui vanno rappresentati i valori numerici, soprattutto per quanto riguarda la separazione tra parte intera e parte decimale.
`LC_TIME`	Categoria che definisce il modo corretto di esprimere le informazioni data-orario.
`LC_MESSAGES`	POSIX: Categoria che definisce il formato dei messaggi emessi per informazioni generali e di quelli diagnostici.

File «regex.h»

Funzione	Descrizione
`int regcomp (regex_t *restrict` ***re***`, const char *restrict` ***regex***`, int` ***cflags***`);`	Compila l'espressione regolare descritta dalla stringa ***regex***, componendo il contenuto della variabile strutturata ***re***, tenendo conto delle opzioni ***cflags***.
`void regfree (regex_t *`***re***`);`	Libera la memoria associata all'espressione regolare compilata nella variabile ***re***.

Funzione	Descrizione
`int regexec (const regex_t *restrict` ***re***`, const char *restrict` ***s***`, size_t` ***n***`, regmatch_t` ***m***`[restrict], int` ***eflags***`);`	Compara l'espressione regolare ***re*** con la stringa ***s***, tenendo conto delle opzioni ***eflags***, immettendo le sottostringhe estratte nell'array di stringhe ***m***, sapendo che questo può contenere al massimo ***n*** stringhe.
`size_t regerror (int` ***e***`, const regex_t *restrict` ***re***`, char *restrict` ***t***`, size_t` ***n***`);`	Sulla base del numero di errore ***e*** e dell'espressione regolare ***re***, produce un messaggio di errore nella stringa ***t*** che non può essere più lunga di ***n*** caratteri.

Tipo	Nome	Membri noti del tipo '**`regex_t`**'
`size_t`	`re_sub`	Quantità di sottoespressioni tra parentesi tonde.

Tipo	Nome	Membri noti di una variabile di tipo '**`regmatch_t`**'
`regoff_t`	`rm_so`	Scostamento in byte, dall'inizio della stringa, corrispondente all'inizio della sottostringa individuata.
`regoff_t`	`rm_eo`	Scostamento in byte, dall'inizio della stringa, corrispondente al carattere **successivo** alla sottostringa individuata.

Macro-variabile	Da usare come opzioni della funzione ***regcomp()*** per la compilazione di un'espressione regolare.
`REG_EXTENDED`	L'espressione regolare fornita è di tipo ERE (estesa). Se non si usa questa opzione, si intende che l'espressione sia di tipo BRE.
`REG_ICASE`	L'espressione regolare fornita va valutata senza distinguere tra lettere maiuscole o minuscole.
`REG_NOSUB`	Per la compilazione dell'espressione regolare non si intende tenere conto della corrispondenza eventuale di sottostringhe; in altri termini, non si vogliono considerare le parentesi '**`\(`**' e '**`\)`**', oppure '**`(`**' e '**`)`**' (a seconda che si tratti di ERE o BRE). In tal caso, l'espressione regolare serve per il confronto, ma non per estrapolare porzioni del risultato ottenuto.
`REG_NEWLINE`	In condizioni normali, il codice *new-line* contenuto nella stringa con cui l'espressione regolare deve essere confrontata, viene trattato come gli altri caratteri. Con l'opzione '**`REG_NEWLINE`**', invece, l'operatore '**`^`**' individua l'inizio di un testo che segue un codice *new-line*, mentre l'operatore '**`$`**' individua la fine di un testo che precede un codice *new-line*.

Macro-variabile Significato	Da usare come opzioni della funzione ***regexec()*** per la comparazione di un'espressione regolare già compilata con una stringa.
`REG_NOTBOL`	In condizioni normali, il carattere '`^`' trova corrispondenza con l'inizio di una stringa. Con questa opzione, si inibisce tale corrispondenza (*not begin of line*).
`REG_NOTEOL`	In condizioni normali, il carattere '`$`' trova corrispondenza con la fine di una stringa. Con questa opzione, si inibisce tale corrispondenza (*not end of line*).

Macro-variabile	Tipo di errore restituito dalla funzione ***regcomp()*** o da ***regexec()***.
`REG_BADBR`	Il contenuto di '`\{…\}`' (nel caso di BRE) o di '`{…}`' (nel caso di ERE), risulta non valido: potrebbe non trattarsi di un numero, oppure potrebbe esserci un numero troppo grande, oppure potrebbero esserci più di due numeri, oppure il primo potrebbe essere più grande del secondo. Infatti, il contenuto di tale raggruppamento deve essere un numero singolo, oppure due numeri separati da una virgola, dove il primo deve essere inferiore al secondo.
`REG_BADPAT`	Espressione regolare non valida (errore di sintassi).

Macro-variabile	Tipo di errore restituito dalla funzione ***regcomp()*** o da ***regexec()***.
`REG_BADRPT`	Un operatore di ripetizione, del tipo '`?`', '`*`' o '`+`', non è preceduto a un'espressione regolare, ovvero si trova in una posizione sbagliata.
`REG_ECOLLATE`	Elemento di collazione (*collating element*) non valido, nell'ambito della configurazione locale attuale.
`REG_ECTYPE`	Riferimento a una classe di caratteri non valida.
`REG_EESCAPE`	L'espressione regolare termina con '`\`' e ciò non è ammissibile.
`REG_ESUBREG`	Una sequenza '`\`***n***', dove ***n*** è un numero, è errata.
`REG_EBRACE`	Le parentesi graffe che descrivono la ripetizione di qualcosa non bilanciano. Può trattarsi di sequenze del tipo '`\{`…`\}`' per le espressioni BRE o del tipo '`{`…`}`' per le espressioni ERE.
`REG_EBRACK`	Parentesi quadre non bilanciate (parentesi aperta e non chiusa, o viceversa).
`REG_EPAREN`	Le parentesi tonde che descrivono delle sottoespressioni non bilanciano. Può trattarsi di sequenze del tipo '`\(`…`\)`' per le espressioni BRE o del tipo '`(`…`)`' per le espressioni ERE.
`REG_ERANGE`	Un'estremità di un intervallo di valori non è valido.
`REG_ESPACE`	Il procedimento di interpretazione dell'espressione regolare porta all'esaurimento della memoria disponibile.

Macro-variabile	Tipo di errore restituito dalla funzione ***regcomp()*** o da ***regexec()***.
`REG_NOMATCH`	Non c'è corrispondenza tra espressione regolare e stringa.

File «sys/stat.h»

«

Funzione	Descrizione
`int chmod (const char *`***path***`,` `mode_t` ***mode***`);`	Cambia i permessi di un file, individuato dal suo percorso nel file system, rappresentati da una variabile di tipo '**`mode_t`**' (il tipo di file non può essere cambiato).
`int fchmod (int` ***fdn***`,` `mode_t` ***mode***`);`	Cambia i permessi di un file aperto, individuato da un descrittore, rappresentati da una variabile di tipo '**`mode_t`**' (il tipo di file non può essere cambiato).
`int fstat (int` ***fdn***`,` `struct stat *`***buffer***`);`	Aggiorna i membri della struttura a cui punta ***buffer***, con le informazioni relative al file aperto con descrittore ***fdn***.

Funzione	Descrizione
`int lstat (const char *restrict` ***path*****`,`** `struct stat *restrict` ***buffer*****`);`**	Aggiorna i membri della struttura a cui punta ***buffer***, con le informazioni relative al file individuato dal percorso ***path***. Ma a differenza di ***stat()***, se il file richiesto è un collegamento simbolico, si ottengono le informazioni del file che rappresenta il collegamento stesso, ignorando ciò a cui questo punterebbe.
`int mkdir (const char *`***path*****`,`** `mode_t` ***mode*****`);`**	Crea una directory, specificata attraverso un percorso del file system, con i permessi indicati nel parametro ***mode***, dove non si può specificare il tipo di file. I permessi richiesti vengono filtrati dalla maschera dei permessi.
`int mkfifo (const char *`***path*****`,`** `mode_t` ***mode*****`);`**	Crea un file FIFO, individuato dal suo percorso nel file system, utilizzando i permessi richiesti, subordinatamente al filtro della maschera dei permessi.

Funzione	Descrizione
`int mknod (const char *`***path***`,` `mode_t mode, dev_t dev);`	Crea virtualmente un file di qualunque tipo, specificando il percorso nel file system, il tipo di file e i permessi (in questo caso il parametro ***mode*** serve principalmente per specificare il tipo di file) e il numero di dispositivo complessivo, nel caso particolare di un file di dispositivo. In condizioni normali, non dovrebbe essere possibile la creazione di una directory, ma se anche fosse possibile, sarebbe sconsigliabile l'uso di questa funzione per tale scopo.

`int stat`
`(const char *restrict` ***path***`,`
`struct stat *restrict` ***buffer***`);`
	Aggiorna i membri della struttura a cui punta ***buffer***, con le informazioni relative al file individuato dal percorso ***path***.

`mode_t umask (mode_t` ***mask***`);`	Definisce la maschera dei permessi per il processo in corso, restituendo il valore precedente della maschera.

Macroistruzione	Descrizione
`S_ISBLK (`***mode***`)`	Restituisce *Vero* se il parametro ***mode*** rappresenta un file di dispositivo a blocchi.
`S_ISCHR (`***mode***`)`	Restituisce *Vero* se il parametro ***mode*** rappresenta un file di dispositivo a caratteri.
`S_ISFIFO (`***mode***`)`	Restituisce *Vero* se il parametro ***mode*** rappresenta un file FIFO.
`S_ISREG (`***mode***`)`	Restituisce *Vero* se il parametro ***mode*** rappresenta un file puro e semplice.
`S_ISDIR (`***mode***`)`	Restituisce *Vero* se il parametro ***mode*** rappresenta una directory.
`S_ISLNK (`***mode***`)`	Restituisce *Vero* se il parametro ***mode*** rappresenta un collegamento simbolico.
`S_ISSOCK (`***mode***`)`	Restituisce *Vero* se il parametro ***mode*** rappresenta un socket di dominio Unix.

Tipo	Nome	Membri noti del tipo derivato '**`struct stat`**'
`dev_t`	`st_dev`	Numero di dispositivo dell'unità contenente il file.
`ino_t`	`st_ino`	Numero di inode del file.
`mode_t`	`st_mode`	Tipo di file e permessi di accesso relativi.
`nlink_t`	`st_nlink`	Collegamenti riferiti al file.
`uid_t`	`st_uid`	Numero UID dell'utente proprietario del file.

Tipo	Nome	Membri noti del tipo derivato '**struct stat**'
`gid_t`	`st_gid`	Numero GID del gruppo proprietario del file.
`dev_t`	`st_rdev`	Numero di dispositivo rappresentato, nel caso si tratti effettivamente di un file di dispositivo.
`off_t`	`st_size`	Dimensione del file.
`time_t`	`st_atime`	Data e orario dell'ultimo accesso al file.
`time_t`	`st_mtime`	Data e orario dell'ultima modifica apportata al contenuto del file.
`time_t`	`st_ctime`	Data e orario dell'ultima modifica di inode (data di creazione dell'inode).
`blksize_t`	`st_blksize`	Dimensione del blocco per le operazioni di input-output.
`blkcnt_t`	`st_blocks`	Dimensione del file espressa in blocchi.

Macro-variabile	Da usare per la definizione di un tipo di file, in variabili di tipo '**mode_t**'.
`S_IFMT`	Maschera che raccoglie tutti i bit che individuano il tipo di file.
`S_IFBLK`	File di dispositivo a blocchi.
`S_IFCHR`	File di dispositivo a caratteri.
`S_IFIFO`	File FIFO.
`S_IFREG`	File puro e semplice.
`S_IFDIR`	Directory.

Macro-variabile	Da usare per la definizione di un tipo di file, in variabili di tipo '**mode_t**'.
S_IFLNK	Collegamento simbolico.
S_IFSOCK	Socket di dominio Unix.

Macro-variabile	Da usare per la definizione dei permessi di un file, in variabili di tipo '**mode_t**'.
S_ISUID	SUID.
S_ISGID	SGID.
S_ISVTX	Sticky.
S_IRWXU	Lettura, scrittura ed esecuzione per l'utente proprietario.
S_IRUSR	Lettura per l'utente proprietario.
S_IWUSR	Scrittura per l'utente proprietario.
S_IXUSR	Esecuzione per l'utente proprietario.
S_IRWXG	Lettura, scrittura ed esecuzione per il gruppo.
S_IRGRP	Lettura per il gruppo.
S_IWGRP	Scrittura per il gruppo.
S_IXGRP	Esecuzione per il gruppo.
S_IRWXO	Lettura, scrittura ed esecuzione per gli altri utenti.

Macro-variabile	Da usare per la definizione dei permessi di un file, in variabili di tipo '**mode_t**'.
S_IROTH	Lettura per gli altri utenti.
S_IWOTH	Scrittura per gli altri utenti.
S_IXOTH	Esecuzione per gli altri utenti.