

APPENDICE G

La libreria standard

G.1 Errori <errno.h>

EDOM
ERANGE

Questi simboli si espandono in espressioni costanti intere con valori distinti diversi da zero, adatti all'utilizzo nelle direttive **#if** del preprocessore.

`errno`

Un valore di tipo **int** che è impostato con un numero di errore positivo da diverse funzioni della libreria. Il valore di **errno** è zero all'inizio dell'esecuzione del programma, ma non è mai impostato a zero da nessuna funzione della libreria. Un programma che utilizzi **errno** per il controllo dell'errore dovrebbe impostarlo a zero prima della chiamata a una funzione della libreria e controllare il suo valore prima di effettuarne un'altra. All'inizio della propria esecuzione, una funzione della libreria potrebbe salvare il valore di **errno** e impostarlo a zero, ripristinando quello originale, qualora **errno** valga ancora zero poco prima del ritorno. Che ci sia o no un errore, **errno** potrebbe essere impostato a un valore diverso da zero da una chiamata a una funzione della libreria, qualora l'utilizzo di **errno** non sia documentato nella descrizione della funzione riportata dallo standard.

G.2 Definizioni comuni <stddef.h>

NULL

Una costante di puntatore nullo definita dall'implementazione.

`offsetof(tipo, designatore-di-membro)`

Si espande in un'espressione costante intera di tipo **size_t**, il cui valore è l'offset in byte per il membro della struttura (specificato dal designatore-di-membro) dall'inizio del suo tipo di struttura (specificato da `tipo`). Il designatore-di-membro dovrà essere tale che, dato

```
static tipo t;
```

l'espressione **&(t.designatore-di-membro)** sarà valutata in una costante di indirizzo. (Il comportamento sarà indefinito, qualora il membro specificato sia un campo di bit).

`ptrdiff_t`

Il tipo intero con segno restituito dalla sottrazione tra due puntatori.

`size_t`

Il tipo intero senza segno restituito dall'operatore **sizeof**.

wchar_t

Un tipo intero, il cui intervallo di valori può rappresentare codici distinti per tutti i membri dell'insieme di caratteri più grande specificato tra quelli delle localizzazioni supportate. Il carattere nullo deve avere il valore di codice zero, mentre ogni membro dell'insieme di caratteri di base deve averne uno uguale a quello che si otterrebbe utilizzandolo da solo in una costante di carattere intera.

G.3 Diagnostica <assert.h>

```
void assert(int espressione);
```

La macro **assert** inserisce delle istruzioni diagnostiche nei programmi. Qualora **espressione** sia falsa, quando sarà eseguita, la macro **assert** scriverà nel file dello standard error delle informazioni relative alla chiamata fallita (incluso il testo dell'argomento, il nome del file sorgente e il suo numero di riga: gli ultimi sono rispettivamente i valori delle macro del preprocessore **__FILE__** e **__LINE__**), in un formato definito dall'implementazione. Il messaggio scritto potrebbe avere la forma

```
Assertion failed: espressione, file xyz, line nnn
```

La macro **assert** richiamerà quindi la funzione **abort**. Qualora la direttiva del preprocessore

```
#define NDEBUG
```

appaia in un file sorgente in cui sia stato incluso **assert.h**, tutte le asserzioni del file saranno ignorate.

G.4 Gestione dei caratteri <ctype.h>

Le funzioni di questa sezione restituiscono un valore diverso da zero (vero), se e solo se il valore dell'argomento **c** è conforme a quello descritto nel commento della funzione.

```
int isalnum(int c);
```

Verifica qualsiasi carattere per il quale siano vere **isalpha** o **isdigit**.

```
int isalpha(int c);
```

Verifica qualsiasi carattere per il quale siano vere **isupper** o **islower**.

```
int iscntrl(int c);
```

Verifica qualsiasi carattere di controllo.

```
int isdigit(int c);
```

Verifica qualsiasi carattere numerico decimale.

```
int isgraph(int c);
```

Verifica qualsiasi carattere stampabile, eccetto lo spazio (' ').

```
int islower(int c);
```

Verifica qualsiasi carattere che sia una lettera minuscola.

```
int isprint(int c);
```

Verifica qualsiasi carattere stampabile, incluso lo spazio (' ').

```
int ispunct(int c);
```

Verifica ogni carattere stampabile che non sia né uno spazio (' ') né un carattere per il quale sia vera **isalnum**.

```
int isspace(int c);
```

Verifica qualsiasi carattere che corrisponda a uno spazio bianco standard. I caratteri di spazio bianco standard sono: lo spazio (' '), il salto pagina ('\f'), il newline ('\n'), il ritorno carrello ('\r'), la tabulazione orizzontale ('\t') e quella verticale ('\v').

```
int isupper(int c);
```

Verifica qualsiasi carattere che sia una lettera maiuscola.

```
int isxdigit(int c);
```

Verifica qualsiasi carattere che sia un numero esadecimale.

```
int tolower(int c);
```

Converte una lettera maiuscola nella sua corrispondente minuscola. Se la funzione **isupper**, applicata all'argomento, è vera ed esiste un carattere corrispondente per cui la funzione **islower** sia vera, allora la funzione **tolower** restituirà quest'ultimo, in caso contrario verrà restituito l'argomento inalterato.

```
int toupper(int c);
```

Converte una lettera minuscola nella sua corrispondente maiuscola. Se la funzione **islower**, applicata all'argomento, è vera ed esiste un carattere corrispondente per cui la funzione **isupper** sia vera, allora la funzione **toupper** restituirà quest'ultimo, in caso contrario verrà restituito l'argomento inalterato.

G.5 Localizzazione <locale.h>

```
LC_ALL
```

```
LC_COLLATE
```

```
LC_CTYPE
```

```
LC_MONETARY
```

```
LC_NUMERIC
```

```
LC_TIME
```

Questi simboli si espandono in espressioni costanti intere con valori distinti, adatti a essere utilizzati come primo argomento della funzione **setlocale**.

```
NULL
```

Una costante di puntatore nullo definita dall'implementazione.

```
struct lconv
```

Contiene dei membri correlati con la formattazione dei valori numerici. La struttura deve contenere in qualsiasi ordine almeno i seguenti membri. Nella localizzazione **"C"**, i membri devono avere i valori specificati nei commenti.

```
char *decimal_point;          /* "." */
char *thousands_sep;        /* "" */
char *grouping;              /* "" */
char *int_curr_symbol;       /* "" */
char *currency_symbol;       /* "" */
char *mon_decimal_point;     /* "" */
char *mon_thousands_sep;    /* «» */
char *mon_grouping;          /* "" */
char *positive_sign;         /* «» */
char *negative_sign;         /* «» */
char int_frac_digits;        /* CHAR_MAX */
char frac_digits;            /* CHAR_MAX */
char p_cs_precedes;          /* CHAR_MAX */
char p_sep_by_space;         /* CHAR_MAX */
char n_cs_precedes;          /* CHAR_MAX */
char n_sep_by_space;         /* CHAR_MAX */
char p_sign_posn;            /* CHAR_MAX */
char n_sign_posn;            /* CHAR_MAX */
char *setlocale(int categoria, const char *localizzazione);
```

La funzione **setlocale** seleziona la porzione appropriata della localizzazione del programma, come specificato dagli argomenti **categoria** e **localizzazione**. La funzione **setlocale** può essere utilizzata per modificare o interrogare l'intera localizzazione corrente del programma o porzioni di quella. Il valore **LC_ALL** per **categoria** nomina l'intera localizzazione del programma; gli altri valori per **categoria** nominano soltanto una porzione della localizzazione del programma. **LC_COLLATE** influisce sul comportamento delle funzioni **strcoll** e **strxfrm**. **LC_CTYPE** influenza quello delle funzioni per la gestione dei caratteri e di quelle multi-byte. **LC_MONETARY** agisce sulle informazioni relative alla formattazione dei valori monetari restituite dalla funzione **localeconv**. **LC_NUMERIC** agisce sul carattere separatore dei decimali per le funzioni di formattazione dell'input/output, per quelle di conversione delle stringhe e per l'informazione di formattazione dei valori non monetari restituita da **localeconv**. **LC_TIME** influisce sul comportamento di **strftime**.

Un valore **"C"** per **localizzazione** specifica l'ambiente minimo per la traduzione **C**; mentre il valore **" "** specifica l'ambiente nativo definito dall'implementazione. Possono essere passate a **setlocale** anche altre stringhe definite dall'implementazione. All'inizio dell'esecuzione del programma sarà eseguita l'equivalente di

```
setlocale(LC_ALL, "C");
```

Qualora per **localizzazione** sia stato fornito un puntatore a una stringa e la selezione possa essere soddisfatta, la funzione **setlocale** restituirà un puntatore alla stringa associata alla **categoria** specificata per la nuova localizzazione. Qualora la selezione non possa essere soddisfatta, la funzione **setlocale** restituirà un puntatore nullo e la localizzazione del programma non sarà cambiata.

Un puntatore nullo per **localizzazione** farà sì che la funzione **setlocale** restituisca un puntatore alla stringa associata alla **categoria** per la localizzazione corrente del programma; questa non sarà modificata.

Il puntatore a stringa restituito dalla funzione **setlocale** è tale che una successiva chiamata con quel valore di stringa e la categoria associata ripristinerà quella parte della localizzazione del programma. La stringa puntata sarà modificabile dal programma, ma potrà essere sostituita da una successiva chiamata alla funzione **setlocale**.

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

La funzione **localeconv** imposta i componenti di un oggetto di tipo **struct lconv** con i valori appropriati per la formattazione di quantità numeriche (monetarie e non) in accordo con le regole della localizzazione corrente.

I membri di tipo **char *** della struttura sono puntatori a stringhe, ognuno dei quali (eccetto **decimal_point**) può puntare a "" per indicare che il valore non è disponibile nella localizzazione corrente o è di lunghezza zero. I membri di tipo **char** sono dei numeri non negativi, ognuno dei quali può essere **CHAR_MAX** per indicare che il valore non è disponibile nella localizzazione corrente. I membri inclusi sono:

```
char *decimal_point
```

Il carattere separatore dei decimali, utilizzato per formattare le quantità non monetarie.

```
char *thousands_sep
```

Il carattere utilizzato per separare i gruppi di cifre, prima del carattere separatore dei decimali nelle quantità non monetarie formattate.

```
char *grouping
```

Una stringa i cui elementi indicano la dimensione di ogni gruppo di cifre nelle quantità non monetarie formattate.

```
char *int_curr_symbol
```

Il simbolo internazionale della valuta applicabile alla localizzazione corrente. I primi tre caratteri contengono il simbolo alfabetico internazionale della valuta in accordo con quelli specificati nell'ISO 4217:1987. Il quarto carattere (immediatamente precedente quello nullo) è quello utilizzato per separare il simbolo internazionale della valuta dalla quantità monetaria.

```
char *currency_symbol
```

Il simbolo della valuta della localizzazione applicabile a quella corrente.

```
char *mon_decimal_point
```

Il carattere separatore dei decimali utilizzato per formattare le quantità monetarie.

```
char *mon_thousands_sep
```

Il separatore per i gruppi di cifre prima del carattere separatore dei decimali nelle quantità monetarie formattate.

```
char *mon_grouping
```

Una stringa i cui elementi indicano la dimensione di ogni gruppo di cifre nelle quantità monetarie formattate.

```
char *positive_sign
```

La stringa utilizzata per indicare una quantità monetaria formattata di valore non negativo.

```
char *negative_sign
```

La stringa utilizzata per indicare una quantità monetaria formattata di valore negativo.

```
char int_frac_digits
```

Il numero di cifre frazionarie (quelle dopo il carattere separatore dei decimali) da visualizzare in una quantità monetaria formattata secondo le regole internazionali.

```
char frac_digits
```

Il numero di cifre frazionarie (quelle dopo il carattere separatore dei decimali) da visualizzare in una quantità monetaria formattata.

```
char p_cs_precedes
```

Impostata a 1 o a 0 fa rispettivamente in modo che **currency_symbol** preceda o segua il valore di una quantità monetaria non negativa formattata.

```
char p_sep_by_space
```

Impostata a 1 o a 0 fa rispettivamente in modo che **currency_symbol** sia o no separato con uno spazio dal valore di una quantità monetaria non negativa formattata.

```
char n_cs_precedes
```

Impostata a 1 o a 0 fa rispettivamente in modo che **currency_symbol** preceda o segua il valore di una quantità monetaria negativa formattata.

```
char n_sep_by_space
```

Impostata a 1 o a 0 fa rispettivamente in modo che **currency_symbol** sia o no separato con uno spazio dal valore di una quantità monetaria negativa formattata.

```
char p_sign_posn
```

Impostata a un valore indicante la posizione di **positive_sign** per una quantità monetaria non negativa formattata.

```
char n_sign_posn
```

Impostata a un valore indicante la posizione di **negative_sign** per una quantità monetaria negativa formattata.

Gli elementi di **grouping** e **mon_grouping** sono interpretati in accordo con le seguenti regole:

CHAR_MAX Non deve essere eseguito alcun ulteriore raggruppamento.

0 L'elemento precedente deve essere utilizzato ripetutamente per il resto delle cifre.

altro Il valore intero è il numero di cifre che comprende il gruppo corrente. L'elemento successivo è esaminato per determinare la dimensione del prossimo gruppo di cifre prima di quello corrente.

I valori di **p_sign_posn** e **n_sign_posn** sono interpretati in accordo con le seguenti regole:

- 0 Parentesi intorno alla quantità e a **currency_symbol**.
- 1 La stringa del segno precede la quantità e il **currency_symbol**.
- 2 La stringa del segno segue la quantità e il **currency_symbol**.
- 3 La stringa del segno precede direttamente il **currency_symbol**.
- 4 La stringa del segno segue direttamente il **currency_symbol**.

La funzione **localeconv** restituisce un puntatore all'oggetto completato. La struttura puntata dal valore restituito non dovrebbe essere modificata dal programma, ma può essere sostituita da una chiamata susseguente alla funzione **localeconv**. Inoltre, le chiamate alla funzione **setlocale** con le categorie **LC_ALL**, **LC_MONETARY** o **LC_NUMERIC** possono sostituire i contenuti della struttura.

G.6 Matematica <math.h>

HUGE_VAL

Una costante simbolica rappresentante un'espressione **double** positiva.

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

Calcola il valore principale dell'arcocoseno di **x**. Un errore di dominio si verifica con argomenti non compresi nell'intervallo [-1, +1]. La funzione **acos** restituisce l'arcocoseno nell'intervallo [0, p] radianti.

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

Calcola il valore principale dell'arcoseno di **x**. Un errore di dominio si verifica con argomenti non compresi nell'intervallo [-1, +1]. La funzione **asin** restituisce l'arcoseno nell'intervallo [-p/2, +p/2] radianti.

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

Calcola il valore principale dell'arcotangente di **x**. La funzione **atan** restituisce l'arcotangente nell'intervallo [-p/2, +p/2] radianti.

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

La funzione **atan2** calcola il valore principale dell'arcotangente di **y/x**, usando i segni di entrambi gli argomenti per determinare il quadrante del valore restituito. Un errore di dominio può verificarsi se entrambi gli argomenti sono uguali a zero. La funzione **atan2** restituisce l'arcotangente di **y/x** nell'intervallo [-p, +p] radianti.

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

Calcola il coseno di **x** (misurato in radianti).

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

Calcola il seno di **x** (misurato in radianti).

```
double tan(double x);
```

Restituisce la tangente di **x** (misurato in radianti).

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

Calcola il coseno iperbolico di **x**. Si verifica un errore di overflow se il valore assoluto di **x** è troppo grande.

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

Calcola il seno iperbolico di **x**. Si verifica un errore di overflow se il valore assoluto di **x** è troppo grande.

```
double tanh(double x);
```

La funzione **tanh** calcola la tangente iperbolica di **x**.

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

Calcola la funzione esponenziale di **x**. Si verifica un errore di overflow o di underflow se **x** è troppo grande o troppo piccolo.

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

Suddivide il numero in virgola mobile in una frazione normalizzata e in una potenza intera di 2. Immagazzina l'intero nell'oggetto **int** puntato da **exp**. La funzione **frexp** restituisce il valore **x**, tale che **x** è un **double** con grandezza compresa nell'intervallo $[1/2, 1]$ o zero, mentre **valore** è uguale a **x** volte 2 elevato alla potenza ***exp**. Qualora **valore** sia zero, entrambe le parti del risultato saranno zero.

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

Moltiplica un numero in virgola mobile per una potenza intera di 2. Può verificarsi un errore di overflow o di underflow. La funzione **ldexp** restituisce il valore di **x** volte 2 elevato alla potenza **exp**.

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

Calcola il logaritmo naturale di **x**. Un errore di dominio si verifica se l'argomento è negativo. Un errore di overflow può verificarsi se l'argomento è uguale a zero.

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

Calcola il logaritmo in base 10 di **x**. Un errore di dominio si verifica se l'argomento è negativo. Un errore di overflow può verificarsi se l'argomento è uguale a zero.

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

Suddivide l'argomento **valore** in una parte intera e in una frazionaria, ognuna delle quali ha lo stesso segno dell'argomento. Immagazzina la parte intera come **double** nell'oggetto puntato da **iptr**. La funzione **modf** restituisce la parte frazionaria con segno di **valore**.

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

Calcola x elevato alla potenza y . Un errore di dominio si verifica se x è negativo e y non è un valore intero. Un errore di dominio si verifica anche se il risultato non può essere rappresentato, quando x è uguale a zero e y è minore o uguale a zero. Può verificarsi un errore di overflow o di underflow.

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

Calcola la radice quadrata non negativa di x . Un errore di dominio si verifica se l'argomento è negativo.

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

Calcola il valore intero più piccolo non minore di x .

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

Calcola il valore assoluto del numero in virgola mobile x .

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

Calcola il valore intero più grande non maggiore di x .

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

Calcola il resto in virgola mobile di x/y .

G.7 Salti non locali <setjmp.h>

```
jmp_buf
```

Un tipo vettoriale adatto a mantenere le informazioni necessarie per ripristinare un ambiente chiamante.

```
int setjmp(jmp_buf amb);
```

Salva il suo ambiente chiamante nell'argomento di tipo `jmp_buf`, per un successivo utilizzo da parte della funzione `longjmp`.

Nel caso in cui il ritorno fosse da un'invocazione diretta, la macro `setjmp` restituirebbe il valore zero. Nel caso in cui il ritorno fosse da un'invocazione alla funzione `longjmp`, la macro `setjmp` restituirebbe un valore diverso da zero.

Un'invocazione della macro `setjmp` dovrà apparire solamente in uno dei seguenti contesti:

- l'intera espressione di controllo in un'istruzione di selezione o di iterazione;
- un operando di un operatore relazionale o di uguaglianza il cui secondo operando sia un'espressione intera costante e quella risultante sia l'intera espressione di controllo di un'istruzione di selezione o di iterazione;
- l'operando di un operatore unario `!` la cui espressione risultante sia l'intera espressione di controllo di un'istruzione di selezione o di iterazione; o
- l'intera espressione di un'istruzione costituita da una sola espressione.

```
void longjmp(jmp_buf amb, int val);
```

Ripristina l'ambiente salvato dalla più recente invocazione della macro `setjmp` nella stessa chiamata del programma, con il corrispondente argomento di tipo `jmp_buf`. Nel caso in

cui non ci fosse stata una tale invocazione, o se la funzione contenente quella della macro **setjmp** avesse nel frattempo terminato la propria esecuzione, il comportamento sarebbe indefinito.

Tutti gli oggetti avranno i valori che avevano nel momento in cui **longjmp** è stata invocata, eccetto quelli degli oggetti con permanenza automatica locali alla funzione contenente l'invocazione della macro **setjmp** corrispondente, che non siano di tipo volatile e che siano stati cambiati tra la chiamata di **setjmp** e quella di **longjmp**; questi ultimi infatti saranno indeterminati.

Poiché aggira il meccanismo usuale di chiamata e ritorno da funzione, **longjmp** dovrà comportarsi correttamente rispetto agli interrupt, ai segnali e ad ogni loro funzione associata. Il comportamento sarebbe tuttavia indefinito, nel caso in cui la funzione **longjmp** dovesse essere invocata da un gestore di segnali nidificato (cioè, da una funzione invocata come risultato di un segnale sollevato durante la gestione di un altro segnale).

Dopo che **longjmp** sarà stata completata, l'esecuzione del programma continuerà come se la corrispondente invocazione della macro **setjmp** avesse appena restituito il valore specificato da **val**. La funzione **longjmp** non potrà fare in modo che la macro **setjmp** restituisca il valore 0; se **val** dovesse essere 0, la macro **setjmp** restituirebbe il valore 1.

G.8 Gestione dei segnali <signal.h>

`sig_atomic_t`

Il tipo intero di un oggetto a cui si può accedere come un'entità atomica, anche in presenza di interrupt asincroni.

`SIG_DFL`
`SIG_ERR`
`SIG_IGN`

Questi simboli si espandono in espressioni costanti con valori distinti, che hanno un tipo compatibile con il secondo argomento e il valore di ritorno della funzione **signal**, e il cui valore è diverso dall'indirizzo di ogni funzione dichiarabile e dalle seguenti costanti, che si espandono in espressioni costanti positive intere corrispondenti al numero del segnale per la condizione specificata:

SIGABRT	terminazione anormale, come quella che è avviata dalla funzione abort .
SIGFPE	un'operazione aritmetica errata, come una divisione per zero o un'operazione risultante in un overflow.
SIGILL	individuazione di un'immagine di funzione non valida, come un'istruzione illegale.
SIGINT	ricezione di un segnale di attenzione interattivo.
SIGSEGV	un accesso non valido alla memoria.
SIGTERM	una richiesta di chiusura inviata al programma.

Un'implementazione non ha la necessità di generare nessuno di questi segnali, eccetto che come risultato di chiamate esplicite alla funzione **raise**.

```
void (*signal (int seg, void (*funz)(int)))(int);
```

Sceglie uno dei tre modi nei quali sarà gestito il ricevimento del numero di segnale **seg**. Nel caso in cui il valore di **funz** fosse **SIG_DEF**, per quel segnale sarebbe utilizzata la gestione di default. Nel caso in cui il valore di **funz** fosse **SIG_IGN**, il segnale sarebbe ignorato. Altrimenti, **funz** dovrebbe puntare a una funzione da richiamare qualora il segnale si dovesse presentare. Una tale funzione è detta gestore del segnale.

Qualora **funz** punti a una funzione, quando si presenterà il segnale sarà eseguito per primo l'equivalente di **signal(seg, SIG_DFL)**; o un'intercettazione del segnale definita dall'implementazione. (Qualora il valore di **seg** sia **SIGILL** e l'implementazione lo preveda, **SIG_DFL** sarà ripristinato). In seguito sarà eseguito l'equivalente di **(*funz)(seg)**; . La funzione **funz** potrebbe terminare eseguendo un'istruzione **return** o richiamando le funzioni **abort**, **exit** o **longjmp**. Qualora **funz** esegua un'istruzione **return** e il valore di **seg** sia **SIGFPE** o qualsiasi altro valore corrispondente a un'eccezione di calcolo, secondo quanto definito dall'implementazione, il comportamento sarà indefinito. Altrimenti, il programma riprenderà l'esecuzione dal punto in cui era stato interrotto.

Qualora si presentasse un segnale diverso da quelli risultanti da chiamate alle funzioni **abort** o **raise**, il comportamento sarebbe indefinito, nel caso che il gestore del segnale richiamasse una funzione della libreria standard che non sia **signal**, passando come primo argomento il numero del segnale corrispondente a quello che ha causato l'invocazione del gestore, o facesse un riferimento a un oggetto statico che non sia l'assegnazione di un valore a una variabile statica del tipo **volatile sig_atomic_t**. Inoltre, se una tale chiamata alla funzione **signal** dovesse risultare in un ritorno **SIG_ERR**, il valore di **errno** sarebbe indeterminato.

All'inizio dell'esecuzione del programma, potrebbe essere eseguito l'equivalente di

```
signal(seg, SIG_IGN);
```

per alcuni segnali selezionati in un modo definito dall'implementazione; mentre l'equivalente di

```
signal(seg, SIG_DFL);
```

sarà eseguito per tutti gli altri segnali definiti dall'implementazione.

Qualora la richiesta possa essere soddisfatta, la funzione **signal** restituirà il valore di **funz** per la chiamata a **signal** più recente per il segnale **seg** specificato. Altrimenti, sarà restituito il valore **SIG_ERR** e in **errno** sarà immagazzinato un valore positivo.

```
int raise(int seg);
```

La funzione **raise** invia il segnale **seg** al programma in esecuzione. La funzione **raise** restituisce zero in caso di successo e un valore diverso da zero in caso di fallimento.

G.9 Argomenti variabili <stdarg.h>

va_list

Un tipo adatto a mantenere le informazioni necessarie per le macro **va_start**, **va_arg** e **va_end**. Qualora si desideri un accesso ad argomenti variabili, la funzione chiamata dovrà dichiarare un oggetto (chiamato **ap** in questa sezione) di tipo **va_list**. L'oggetto **ap** potrà essere passato come argomento a un'altra funzione; qualora questa invochi la macro **va_arg**

con il parametro **ap**, il valore di questo nella funzione chiamante sarà determinato e dovrà essere passato alla macro **va_end**, prima di ogni altro riferimento ad **ap**.

```
void va_start(va_list ap, parmN);
```

Dovrà essere invocata prima di qualsiasi accesso agli argomenti senza nome. La macro **va_start** inizializza **ap** perché **va_arg** e **va_end** possano successivamente utilizzarlo. Il parametro **parmN** è l'identificatore di quello più a destra nell'elenco variabile di parametri tra quelli inclusi nella definizione di funzione (quello subito prima di **,** ...). Qualora il parametro **parmN** sia dichiarato con la classe di memoria **register**, con un tipo funzionale o vettoriale, o con uno che non sia compatibile con quello che risulterebbe dopo l'applicazione delle promozioni di default all'argomento, il comportamento sarà indefinito.

```
tipo va_arg(va_list ap, tipo);
```

Si espande in un'espressione che ha il tipo e il valore dell'argomento successivo nella chiamata. Il parametro **ap** dovrà essere lo stesso **va_list ap** inizializzato da **va_start**. Ogni invocazione di **va_arg** modificherà **ap** in modo che siano restituiti a turno i valori degli argomenti successivi. Il parametro **tipo** è un nome di tipo specificato in modo tale che quello di un puntatore a un oggetto del tipo specificato possa essere ottenuto semplicemente aggiungendo il suffisso ***** a tipo. Qualora non ci sia nessun argomento successivo, o qualora il tipo non sia compatibile con quello dell'argomento successivo (una volta promosso in base alle promozioni di default), il comportamento sarà indefinito. La prima invocazione della macro **va_arg**, dopo quella di **va_start**, restituirà il valore dell'argomento successivo a quello specificato da **parmN**. Le invocazioni susseguenti restituiranno in successione i valori degli argomenti rimanenti.

```
void va_end(va_list ap);
```

Facilita un normale ritorno da una funzione il cui elenco variabile di argomenti sia stato oggetto di riferimento da parte dell'espansione con cui **va_start** ha inizializzato **va_list ap**. La macro **va_end** può modificare **ap** in modo che non sia più utilizzabile (senza l'intervento di una chiamata a **va_start**). Qualora non ci sia un'invocazione corrispondente della macro **va_start**, o qualora **va_end** non sia stata invocata prima del ritorno, il comportamento sarà indefinito.

G.10 Input/Output <stdio.h>

```
_IOFBF
_IOLBF
_IONBF
```

Espressioni costanti intere con valori distinti, adatte all'utilizzo come terzo argomento della funzione **setvbuf**.

```
BUFSIZ
```

Un'espressione costante intera, che rappresenta la dimensione del buffer utilizzato dalla funzione **setbuf**.

```
EOF
```

Un'espressione costante intera negativa restituita da diverse funzioni per indicare la fine del file, ovverosia, che non c'è più alcun input nello stream.

FILE

Un tipo di oggetto in grado di registrare tutte le informazioni necessarie per controllare uno stream, incluso il suo indicatore di posizione del file, un puntatore al buffer associato (qualora ce ne sia uno), un indicatore di errore che registri se si verificano errori in lettura/scrittura, e un indicatore di fine del file che registri se sia stata raggiunta la fine del medesimo.

FILENAME_MAX

Un'espressione costante intera corrispondente alla dimensione di un vettore di tipo **char**, che sia sufficientemente grande per contenere la stringa del nome di file più lungo che l'implementazione garantisca di poter aprire.

FOPEN_MAX

Un'espressione costante intera corrispondente al numero minimo di file che l'implementazione garantisca di poter aprire contemporaneamente.

fpos_t

Un tipo di oggetto in grado di registrare tutte le informazioni necessarie per specificare in modo univoco ogni posizione all'interno di un file.

L_tmpnam

Un'espressione costante intera corrispondente alla dimensione di un vettore di tipo **char**, che sia sufficientemente grande per contenere la stringa del nome di un file temporaneo generata dalla funzione **tmpnam**.

NULL

Una costante di tipo puntatore nullo definita dall'implementazione.

SEEK_CUR

SEEK_END

SEEK_SET

Espressioni costanti intere con valori distinti, adatte all'utilizzo come terzo argomento della funzione **fseek**.

size_t

Il tipo intero senza segno restituito dall'operatore **sizeof**.

stderr

Espressione di tipo "puntatore a **FILE**" che fa riferimento all'oggetto **FILE** associato allo stream dello standard error.

stdin

Espressione di tipo "puntatore a **FILE**" che fa riferimento all'oggetto **FILE** associato allo stream dello standard input.

stdout

Espressione di tipo "puntatore a **FILE**" che fa riferimento all'oggetto **FILE** associato allo stream dello standard output.

TMP_MAX

Un'espressione costante intera corrispondente al numero minimo di nomi di file univoci che possano essere generati dalla funzione **tmpnam**. Il valore della macro **TMP_MAX** deve essere almeno 25.

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

Fa in modo che il file il cui nome è dato dalla stringa puntata da **nomefile** non sia più accessibile con quel nome. Un tentativo susseguente di aprire quel file utilizzando quel nome fallirà, a meno che non sia stato creato nuovamente. Il comportamento della funzione **remove**, nel caso che il file sia correntemente aperto, dipende dall'implementazione. La funzione **remove** restituirà zero qualora l'operazione abbia successo, un valore diverso da zero qualora fallisca.

```
int rename(const char *vecchio, const char *nuovo);
```

Fa in modo che il file il cui nome corrisponde alla stringa puntata da **vecchio** sia rinominato, da questo momento in poi, con il nome contenuto nella stringa puntata da **nuovo**. Il file chiamato **vecchio** non sarà più accessibile con quel nome. Il comportamento della funzione **rename**, nel caso che un file chiamato con la stringa puntata da **nuovo** esista già prima della sua invocazione, dipenderà dall'implementazione. La funzione **rename** restituirà zero qualora l'operazione abbia successo, un valore diverso da zero qualora fallisca; in questo caso e qualora il file esistesse già, conserverebbe ancora il suo nome originale.

```
FILE *tmpfile(void);
```

Crea un file binario temporaneo che sarà rimosso automaticamente alla sua chiusura o alla fine del programma. L'eventuale rimozione di un file temporaneo aperto, nel caso che il programma termini in modo anormale, dipenderà dall'implementazione. Il file sarà aperto per l'aggiornamento con il modo "**wb+**". La funzione **tmpfile** restituirà un puntatore allo stream del file che sarà stato creato. Qualora il file non possa essere creato, la funzione **tmpfile** restituirà un puntatore nullo.

```
char *tmpnam(char *s);
```

La funzione **tmpnam** genera una stringa che sarà un nome di file valido e che sarà diverso da quelli di qualsiasi file esistente. La funzione **tmpnam** genererà una stringa differente ogni volta che sarà richiamata, fino a un massimo di **TMP_MAX** volte. Il suo comportamento, qualora sia richiamata più di **TMP_MAX** volte, dipenderà dall'implementazione.

Qualora l'argomento sia un puntatore nullo, la funzione **tmpnam** lascerà il suo risultato in un oggetto statico interno e ne restituirà un puntatore. Le successive chiamate della funzione **tmpnam** potranno modificare lo stesso oggetto. Qualora l'argomento non sia un puntatore nullo, si assumerà che punti a un vettore di almeno **L_tmpnam** caratteri; la funzione **tmpnam** scriverà il proprio risultato in quel vettore e restituirà l'argomento.

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

La funzione **fclose** fa in modo che lo stream puntato da **stream** sia svuotato e che il file associato sia chiuso. Ogni dato presente nel buffer e non ancora scritto per quello stream sarà inviato all'ambiente di esecuzione perché sia scritto nel file; ogni dato presente nel buffer e non ancora letto sarà dimenticato. Lo stream sarà dissociato dal file. Qualora il buffer associa-

to sia stato allocato in modo automatico, sarà rilasciato. La funzione **fclose** restituirà zero qualora lo stream sia stato chiuso con successo, o **EOF** nel caso che sia stato rilevato un errore qualsiasi.

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

Nel caso che **stream** punti a uno stream di output o di aggiornamento in cui l'operazione più recente non sia stata una di input, la funzione **fflush** farà in modo che i dati non ancora scritti per quello stream siano inviati all'ambiente di esecuzione perché siano scritti nel file; altrimenti, il comportamento sarà indefinito.

Nel caso che **stream** sia un puntatore nullo, la funzione **fflush** eseguirà la suddetta azione di svuotamento su tutti gli stream che abbiano le suddette caratteristiche. La funzione **fflush** restituirà **EOF** nel caso che si verifichi un errore di scrittura, zero in caso contrario.

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

La funzione **fopen** apre il file il cui nome corrisponde alla stringa puntata da **nomefile** e vi associa uno stream. L'argomento **modo** punta a una stringa che incomincia con una delle seguenti sequenze:

r	apre un file di testo per la lettura
w	tronca la lunghezza a zero o crea un file di testo per la scrittura
a	accodamento; apre o crea un file di testo per scrivere alla fine dello stesso
rb	apre un file binario per la lettura
wb	tronca la lunghezza a zero o crea un file binario per la scrittura
ab	accodamento; apre o crea un file binario per la scrittura alla fine dello stesso
r+	apre un file di testo per l'aggiornamento (lettura e scrittura)
w+	tronca la lunghezza a zero o crea un file di testo per l'aggiornamento
a+	accodamento; apre o crea un file di testo per l'aggiornamento, scrivendo alla fine dello stesso
r+b o rb+	apre un file binario per l'aggiornamento (lettura e scrittura)
w+b o wb+	tronca la lunghezza a zero o crea un file binario per l'aggiornamento
a+b o ab+	accodamento; apre o crea un file binario per l'aggiornamento, scrivendo alla fine dello stesso

L'apertura di un file con il modo di lettura ('**r**' come primo carattere nell'argomento **modo**) fallirà qualora il file non esista o non possa essere letto. L'apertura del file con il modo di accodamento ('**a**' come primo carattere nell'argomento **modo**) farà in modo che tutte le successive scritture nel file siano forzate alla sua fine corrente, ignorando eventuali chiamate alla funzione **fseek**. In alcune implementazioni, aprire un file binario con il modo di accodamento ('**b**' come secondo o terzo carattere nella succitata lista di valori per l'argomento **modo**) potrà inizialmente posizionare l'indicatore per lo stream oltre l'ultimo dato scritto, a causa del riempimento dello spazio rimanente con caratteri nulli.

Quando un file sarà aperto con il modo di aggiornamento ('**+**' come secondo o terzo carattere nella succitata lista di valori per l'argomento **modo**), sullo stream associato potranno essere eseguite delle operazioni di input e di output. Tuttavia, l'output potrebbe non essere seguito direttamente da un input senza la frapposizione di una chiamata alla funzione **fflush**

o a quelle di posizionamento nel file (**fseek**, **fsetpos** o **rewind**), e l'input potrebbe non essere seguito direttamente dall'output senza la frapposizione di una chiamata alle funzioni di posizionamento nel file, sempre che l'operazione di input non incontri la fine del file. In alcune implementazioni, aprire (o creare) un file di testo con il modo di aggiornamento potrebbe invece aprire (o creare) uno stream binario.

Quando sarà aperto, uno stream sarà gestito completamente con una memoria tampone (buffer) se e solo se potrà essere determinato che non faccia riferimento a un dispositivo interattivo. Gli indicatori di errore e di fine del file per lo stream saranno azzerati. La funzione **fopen** restituirà un puntatore all'oggetto che controllerà lo stream. Qualora l'operazione di apertura fallisse, **fopen** restituirebbe un puntatore nullo.

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

La funzione **freopen** apre il file il cui nome corrisponde alla stringa puntata da **nomefile** e vi associa lo stream puntato da **stream**. L'argomento **modo** è utilizzato proprio come nella funzione **fopen**.

La funzione **freopen** tenterà prima di chiudere qualsiasi file associato allo stream specificato. Un eventuale fallimento della chiusura del file sarà ignorato. Gli indicatori di errore e di fine del file per lo stream saranno azzerati. La funzione **freopen** restituirà un puntatore nullo qualora l'operazione di apertura fallisca. Altrimenti, **freopen** restituirà il valore di **stream**.

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

La funzione **setbuf** è equivalente a **setvbuf** invocata con i valori **_IOFBF** per **modo** e **BUFSIZ** per **dimensione**, o (qualora **buf** sia un puntatore nullo), con il valore **_IONBF** per **modo**. La funzione **setbuf** non restituisce alcun valore.

```
int setvbuf(FILE *stream, char *buf, int modo, size_t dimensione);
```

La funzione **setvbuf** può essere utilizzata soltanto dopo che lo stream puntato da **stream** sarà stato associato a un file aperto, e prima che sia eseguita qualsiasi altra operazione sullo stream. L'argomento **modo** determinerà il modo in cui sarà gestito il buffer di **stream**: **_IOFBF** farà in modo che l'input/output passi totalmente dal buffer; **_IOLBF** farà in modo che l'input/output sia passato nel buffer per righe; **_IONBF** farà in modo che l'input/output non passi da un buffer. Qualora **buf** non sia un puntatore nullo, il vettore cui fa riferimento potrà essere utilizzato in sostituzione del buffer allocato dalla funzione **setvbuf**. L'argomento **dimensione** specifica quella del vettore. Il contenuto del vettore sarà sempre indeterminato. La funzione **setvbuf** restituirà zero in caso di successo, o un valore diverso da zero qualora a **modo** sia stato assegnato un valore non valido, o qualora la richiesta non possa essere soddisfatta.

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

La funzione **fprintf** invia il proprio output nello stream puntato da **stream**, in modo controllato dalla stringa puntata da **formato**, che specificherà in che modo debbano essere convertiti per l'output gli argomenti successivi. Qualora non ci siano argomenti sufficienti per il formato, il comportamento sarà indefinito. Qualora il formato sia stato esaurito mentre rimangono ancora degli argomenti, quelli in eccesso saranno come sempre valutati, ma per il resto saranno ignorati. La funzione **fprintf** restituirà il controllo quando avrà incontrato la fine della stringa del formato. Consultate il Capitolo 9, "L'input/output formattato", per una

descrizione dettagliata delle specifiche per la conversione dell'output. La funzione **fprintf** restituirà il numero di caratteri inviati in output, o un valore negativo qualora si verifichi un errore.

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

La funzione **fscanf** legge l'input dallo stream puntato da **stream**, in modo controllato dalla stringa puntata da **formato**, che specificherà le sequenze di input ammissibili e il modo in cui queste dovranno essere convertite per l'assegnamento, utilizzando gli argomenti successivi come puntatori agli oggetti che riceveranno l'input convertito. Qualora non ci siano argomenti sufficienti per il formato, il comportamento sarà indefinito. Qualora il formato sia stato esaurito mentre rimangono ancora degli argomenti, quelli in eccesso saranno come sempre valutati, ma per il resto saranno ignorati. Consultate il Capitolo 9, "L'input/output formattato", per una descrizione dettagliata delle specifiche per la conversione dell'input.

La funzione **fscanf** restituirà il valore della macro **EOF** qualora si verifichi un errore di input prima di qualsiasi conversione. Altrimenti, la funzione **fscanf** restituirà il numero degli elementi assegnati, che potranno anche essere inferiori a quelli forniti, o anche zero, in caso di un prematuro fallimento di corrispondenza.

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

La funzione **printf** è equivalente a **fprintf** con l'argomento **stdout** inserito prima dei rimanenti. La funzione **printf** restituirà il numero di caratteri inviati in output, o un valore negativo qualora si sia verificato un errore.

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

La funzione **scanf** è equivalente a **fscanf** con l'argomento **stdin** inserito prima dei rimanenti. La funzione **scanf** restituirà il valore della macro **EOF**, qualora si sia verificato un errore di input prima di qualsiasi conversione. Altrimenti, **scanf** restituirà il numero degli elementi assegnati, che potranno anche essere inferiori a quelli forniti, o anche zero, in caso di un prematuro fallimento di corrispondenza.

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

La funzione **sprintf** è equivalente a **fprintf**, eccetto che l'argomento **s** specifica un vettore nel quale l'output generato dovrà essere inviato, invece di specificare uno stream. Un carattere nullo sarà accordato alla fine di quelli inviati in output, ma non sarà contato nella somma restituita. Il comportamento della copia fra oggetti che si sovrappongono sarà indefinito. La funzione **sprintf** restituirà il numero di caratteri scritti nel vettore, escluso quello nullo di terminazione.

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

La funzione **sscanf** è equivalente a **fscanf**, eccetto che l'argomento **s** specifica una stringa dalla quale dovrà essere ottenuto l'input, invece di specificare uno stream. Il raggiungimento della fine della stringa sarà equivalente alla fine del file incontrata dalla funzione **fscanf**. Qualora la copia avvenga tra oggetti che si sovrappongono, il comportamento sarà indefinito.

La funzione **sscanf** restituirà il valore della macro **EOF**, qualora si sia verificato un errore di input prima di qualsiasi conversione. Altrimenti, **sscanf** restituirà il numero degli ele-

menti assegnati, che potranno anche essere inferiori a quelli forniti, o anche zero, in caso di un prematuro fallimento di corrispondenza.

```
int vfprintf(FILE *stream, const char *formato, va_list arg);
```

La funzione **vfprintf** è equivalente a una **fprintf**, in cui l'elenco variabile di argomenti sia stato sostituito da un **arg** inizializzato dalla macro **va_start** (e dalle probabili chiamate successive a **va_arg**). La funzione **vfprintf** non invocherà la macro **va_end**. Essa restituirà il numero dei caratteri inviati in output, o un valore negativo qualora si sia verificato un errore.

```
int vprintf(const char *formato, va_list arg);
```

La funzione **vprintf** è equivalente a una **printf**, in cui l'elenco variabile di argomenti sia stato sostituito da un **arg** che dovrebbe essere stato inizializzato dalla macro **va_start** (e dalle probabili chiamate successive a **va_arg**). La funzione **vprintf** non invocherà la macro **va_end**. Essa restituirà il numero dei caratteri inviati in output, o un valore negativo qualora si sia verificato un errore.

```
int vsprintf(char *s, const char *formato, va_list arg);
```

La funzione **vsprintf** è equivalente a una **sprintf**, in cui l'elenco variabile di argomenti sia stato sostituito da un **arg** che dovrebbe essere stato inizializzato dalla macro **va_start** (e dalle probabili chiamate successive a **va_arg**). La funzione **vsprintf** non invocherà la macro **va_end**. Qualora la copia avvenga tra oggetti che si sovrappongano, il comportamento sarà indefinito. La funzione **vsprintf** restituirà il numero dei caratteri scritti nel vettore, escluso quello nullo di terminazione.

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

La funzione **fgetc** ottiene dallo stream di input puntato da **stream** il carattere successivo (se presente), come un **unsigned char** convertito in **int**, e fa avanzare (se definito) l'indicatore di posizione del file associato allo stream. La funzione **fgetc** restituirà il carattere successivo dello stream di input puntato da **stream**. Qualora lo stream sia alla fine del file, sarà impostato il relativo indicatore e **fgetc** restituirà un **EOF**. Qualora si verifichi un errore di lettura, sarà impostato il relativo indicatore dello stream e **fgetc** restituirà un **EOF**.

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

La funzione **fgets** legge dallo stream puntato da **stream** un numero massimo di caratteri inferiore di un'unità a quanto specificato da **n**, immagazzinandoli nel vettore puntato da **s**. Non saranno letti ulteriori caratteri dopo un newline (che sarà conservato), o dopo la fine del file. Immediatamente dopo l'ultimo carattere letto, nel vettore sarà inserito quello nullo di terminazione.

La funzione **fgets** restituirà **s** in caso di successo. Qualora sia stata incontrata la fine del file e nessun carattere sia stato letto e immagazzinato nel vettore, i suoi contenuti resteranno invariati e sarà restituito un puntatore nullo. Qualora durante l'operazione si verifichi un errore di lettura, i contenuti del vettore saranno indeterminati e sarà restituito un puntatore nullo.

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

La funzione **fputc** scrive il carattere specificato da **c** (convertito in un **unsigned char**) nello stream di output puntato da **stream**, inserendolo nel posto individuato dall'indicatore di posizione del file associato allo stream (qualora sia stato definito), e facendo avanzare in modo appropriato il suddetto indicatore. Qualora il file non possa supportare le richieste di posizionamento, o nel caso che lo stream sia stato aperto in accodamento, il carattere sarà accodato allo stream di output. La funzione **fputc** restituirà il carattere scritto. Qualora si verifichi un errore di scrittura, sarà impostato il relativo indicatore dello stream e **fputc** restituirà un **EOF**.

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

La funzione **fputs** scrive la stringa puntata da **s** nello stream puntato da **stream**. Il carattere nullo di terminazione non sarà scritto. La funzione **fputs** restituirà un **EOF** qualora si verifichi un errore di scrittura; altrimenti restituirà un valore non negativo.

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

La funzione **getc** è equivalente a **fgetc** eccetto che, qualora sia stata implementata con una macro, potrebbe valutare **stream** più di una volta. Di conseguenza, l'argomento passato dovrà essere un'espressione che non produca effetti collaterali.

La funzione **getc** restituirà il carattere successivo dello stream di input puntato da **stream**. Qualora questo sia alla fine del file, sarà impostato il relativo indicatore dello stream e **getc** restituirà un **EOF**. Qualora si verifichi un errore di lettura, sarà impostato il relativo indicatore dello stream e **getc** restituirà un **EOF**.

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

La funzione **getchar** è equivalente a un **getc** con l'argomento **stdin**. La funzione **getchar** restituirà il carattere successivo dello stream di input puntato da **stdin**. Qualora questo sia alla fine del file, sarà impostato il relativo indicatore dello stream e **getchar** restituirà un **EOF**. Qualora si verifichi un errore di lettura, sarà impostato il relativo indicatore dello stream e **getchar** restituirà un **EOF**.

```
char *gets(char *s);
```

La funzione **gets** legge dei caratteri dallo stream di input puntato da **stdin** e li immagazzina nel vettore puntato da **s**, finché non incontra la fine del file o non legge un carattere newline. Questi saranno ignorati e nel vettore, immediatamente dopo l'ultimo carattere letto, sarà inserito quello nullo di terminazione. La funzione **gets** restituirà **s** in caso di successo. Qualora sia stata incontrata la fine del file e nessun carattere sia stato letto e immagazzinato nel vettore, i suoi contenuti resteranno invariati e sarà restituito un puntatore nullo. Qualora si verifichi un errore di lettura durante l'operazione, i contenuti del vettore saranno indeterminati e sarà restituito un puntatore nullo.

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

La funzione **putc** è equivalente a **fputc** eccetto che, qualora sia stata implementata con una macro, potrebbe valutare **stream** più di una volta. Di conseguenza l'argomento non dovrà mai essere un'espressione che produca effetti collaterali. La funzione **putc** restituirà il carattere scritto. Qualora si verifichi un errore di scrittura, sarà impostato il relativo indicatore dello stream e **putc** restituirà un **EOF**.

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

La funzione **putchar** è equivalente a un **putc** con il secondo argomento **stdout**. La funzione **putchar** restituirà il carattere scritto. Qualora si verifichi un errore di scrittura, sarà impostato il relativo indicatore dello stream e **putchar** restituirà un **EOF**.

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

La funzione **puts** scrive la stringa puntata da **s** nello stream puntato da **stdout**, accodando all'output un carattere newline. Il carattere nullo di terminazione non sarà scritto. La funzione **puts** restituirà un **EOF** qualora si verifichi un errore di scrittura; altrimenti restituirà un valore non negativo.

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

La funzione **ungetc** rinvia il carattere specificato da **c** (convertito in un **unsigned char**) nello stream di input puntato da **stream**. I caratteri rinvii saranno restituiti in ordine inverso dalle successive letture su quello stream. La frapposizione di una chiamata con successo (sullo stream puntato da **stream**) di una funzione di posizionamento nel file (**fseek**, **fsetpos** o **rewind**) eliminerà tutti i caratteri rinvii allo stream. La memoria esterna corrispondente allo stream resterà immutata.

Il rinvio di un singolo carattere è sempre garantito. Nel caso in cui la funzione **ungetc** sia invocata troppe volte sullo stesso stream, senza frapporre delle operazioni di lettura o di riposizionamento nel file, ci potrebbe essere un fallimento. Se il valore di **c** dovesse essere uguale a **EOF**, la funzione fallirebbe e lo stream di input resterebbe invariato.

Una chiamata con successo alla funzione **ungetc** azzererà l'indicatore di fine file dello stream. Dopo la lettura e l'eliminazione di tutti i caratteri rinvii, il valore dell'indicatore di posizione per il file dello stream dovrebbe corrispondere a quello precedente il rinvio dei caratteri. Per uno stream di testo, il valore del suo indicatore di posizione del file, dopo una chiamata con successo alla funzione **ungetc**, non sarà determinato finché tutti i caratteri rinvii non saranno letti o eliminati. Per uno stream binario, il suo indicatore di posizione del file sarà determinato da ogni chiamata successiva alla funzione **ungetc**; qualora prima di un'invocazione il suo valore sia zero, questo sarà indeterminato dopo la chiamata. La funzione **ungetc** restituirà il carattere rinvio dopo la conversione, o un **EOF** qualora l'operazione fallisca.

```
size_t fread(void *ptr, size_t dimensione, size_t nmemb, FILE *stream);
```

La funzione **fread** legge dallo stream puntato da **stream** un massimo di **nmemb** elementi di dimensione specificata da **dimensione**, immagazzinandoli nel vettore puntato da **ptr**. L'indicatore di posizione del file per lo stream (se definito) sarà fatto avanzare del numero di caratteri letti con successo. Qualora si verifichi un errore, il valore risultante dell'indicatore di posizione del file per lo stream sarà indeterminato. Qualora sia stato letto un elemento parziale, il suo valore sarà indeterminato.

La funzione **fread** restituirà il numero di elementi letti con successo, che potrà anche essere inferiore a **nmemb** qualora sia stato incontrato un errore di lettura o la fine del file. Nel caso che **dimensione** o **nmemb** sia zero, **fread** restituirà zero e i contenuti del vettore e lo stato dello stream resteranno immutati.

```
size_t fwrite(const void *ptr, size_t dimensione, size_t nmemb, FILE *stream);
```

La funzione **fwrite** scrive nello stream puntato da **stream** un massimo di **nmemb** elementi di dimensione specificata da **dimensione**, leggendoli dal vettore puntato da **ptr**. L'indicatore di posizione del file per lo stream (se definito) sarà fatto avanzare del numero di caratteri scritti con successo. Qualora si verifichi un errore, il valore risultante dell'indicatore di posizione del file per lo stream sarà indeterminato. La funzione **fwrite** restituirà il numero di elementi scritti con successo, che potrà essere inferiore a **nmemb** soltanto qualora si sia verificato un errore di scrittura.

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

La funzione **fgetpos** immagazzina nell'oggetto puntato da **pos** il valore corrente dell'indicatore di posizione relativo al file associato allo stream puntato da **stream**. Il valore immagazzinato conterrà delle informazioni non specificate, che potranno essere utilizzate dalla funzione **fsetpos** per riposizionare lo stream sulla posizione che aveva al momento della chiamata di **fgetpos**. In caso di esito positivo, la funzione **fgetpos** restituirà zero mentre, in caso di fallimento, restituirà un valore diverso da zero e immagazzinerà in **errno** un valore positivo definito dall'implementazione.

```
int fseek(FILE *stream, long int offset, int partenza);
```

La funzione **fseek** imposta l'indicatore di posizione del file per lo stream puntato da **stream**. Per uno stream binario, la nuova posizione, misurata in caratteri dall'inizio del file, sarà ottenuta aggiungendo **offset** alla posizione specificata da **partenza**. La posizione specificata corrisponderà all'inizio del file, qualora **partenza** sia **SEEK_SET**, o al valore corrente dell'indicatore di posizione del file, qualora sia **SEEK_CUR**, oppure alla fine dello stesso, qualora sia **SEEK_END**. Uno stream binario non ha bisogno di supportare in modo significativo le chiamate di **fseek** con un valore di **partenza** uguale a **SEEK_END**. Per uno stream di testo, **offset** dovrà essere zero, oppure un valore restituito da una precedente invocazione della funzione **ftell** sullo stesso stream e **partenza** dovrà essere **SEEK_SET**.

Una chiamata con esito positivo alla funzione **fseek** azzererà l'indicatore di fine del file per lo stream, e annullerà ogni effetto di **ungetc** sullo stesso. Dopo un'invocazione di **fseek**, l'operazione successiva su uno stream di aggiornamento potrà essere un input o un output. La funzione **fseek** restituirà un valore diverso da zero soltanto per una richiesta che non possa essere soddisfatta.

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

La funzione **fsetpos** imposta l'indicatore di posizione del file per lo stream puntato da **stream**, usando il valore dell'oggetto indicato da **pos**; tale valore dovrà essere stato ottenuto da una precedente invocazione della funzione **fgetpos** sullo stesso stream. Una chiamata con esito positivo alla funzione **fsetpos** azzererà l'indicatore di fine del file per lo stream, e annullerà ogni effetto di **ungetc** sullo stesso. Dopo un'invocazione di **fsetpos**, l'operazione successiva in uno stream di aggiornamento potrà essere un input o un output. Qualora abbia successo, la funzione **fsetpos** restituirà zero mentre, in caso di fallimento, restituirà un valore diverso da zero e immagazzinerà in **errno** un valore positivo definito dall'implementazione.

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

La funzione **ftell** ottiene il valore corrente dell'indicatore di posizione del file per lo stream puntato da **stream**. Per uno stream binario, il valore corrisponderà al numero di

caratteri dall'inizio del file. Per uno stream di testo, il suo indicatore di posizione del file conterrà informazioni non specificate, che potranno essere utilizzate dalla funzione **fseek** per far ritornare l'indicatore di posizione del file per lo stream nel punto in cui si trovava al momento della chiamata a **ftell**; la differenza tra i due valori restituiti non è necessariamente una misura indicativa del numero di caratteri scritti o letti. In caso di successo, la funzione **ftell** restituirà il valore dell'indicatore di posizione del file per lo stream. In caso di fallimento, la funzione **ftell** restituirà **-1L** e immagazzinerà in **errno** un valore positivo definito dall'implementazione.

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

La funzione **rewind** imposta l'indicatore di posizione del file per lo stream puntato da **stream** all'inizio del file. È equivalente a

```
(void) fseek(stream, 0L, SEEK_SET)
```

eccetto che sarà anche azzerato l'indicatore di errore per lo stream.

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

La funzione **clearerr** azzerava gli indicatori di fine del file e di errore per lo stream puntato da **stream**.

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

La funzione **feof** controlla l'indicatore di fine del file per lo stream puntato da **stream**. La funzione **feof** restituirà un valore diverso da zero se e solo se sia stato impostato l'indicatore di fine del file per **stream**.

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

La funzione **ferror** controlla l'indicatore di errore per lo stream puntato da **stream**. La funzione **ferror** restituirà un valore diverso da zero se e solo se sia stato impostato l'indicatore di errore per **stream**.

```
void perror(const char *s);
```

La funzione **perror** rileva il messaggio di errore corrispondente al numero presente nell'espressione intera **errno**. Essa scrive la seguente sequenza di caratteri nello stream dello standard error: in primo luogo (qualora il puntatore **s** e il carattere cui fa riferimento non siano nulli), la stringa puntata da **s** seguita dai due punti (:) e da uno spazio; quindi la stringa del messaggio di errore appropriata, seguita da un carattere newline. I contenuti delle stringhe del messaggio di errore saranno identici a quelli restituiti dalla funzione **strerror** con l'argomento **errno** e dipenderanno dall'implementazione.

G.11 Utilità generiche <stdlib.h>

```
EXIT_FAILURE
EXIT_SUCCESS
```

Espressioni intere che possono essere utilizzate come argomenti della funzione **exit** per restituire all'ambiente di esecuzione uno stato di chiusura con esito rispettivamente negativo o positivo.

MB_CUR_MAX

Un'espressione intera positiva, il cui valore corrisponde al numero massimo di byte contenuto in un carattere multibyte dell'insieme esteso, specificato dalla localizzazione corrente (categoria **LC_CTYPE**) e il cui valore non sarà mai maggiore di **MB_LEN_MAX**.

NULL

Una costante di tipo puntatore nullo definita dall'implementazione.

RAND_MAX

Un'espressione costante di intervallo, il cui valore corrisponde a quello massimo restituito dalla funzione **rand**. Il valore della macro **RAND_MAX** dovrà essere almeno 32767.

div_t

Un tipo di struttura corrispondente a quello del valore restituito dalla funzione **div**.

ldiv_t

Un tipo di struttura corrispondente a quello del valore restituito dalla funzione **ldiv**.

size_t

Il tipo intero senza segno restituito dall'operatore **sizeof**.

wchar_t

Un tipo intero, il cui intervallo di valori può rappresentare codici distinti per tutti i membri dell'insieme di caratteri più grande specificato tra quelli delle localizzazioni supportate. Il carattere nullo deve avere il valore di codice zero, mentre ogni membro dell'insieme di caratteri di base deve averne uno uguale a quello che si otterrebbe utilizzandolo da solo in una costante di carattere intera.

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

Converte la porzione iniziale della stringa puntata da **nptr** in una rappresentazione **double**. La funzione **atof** restituirà il valore convertito.

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

Converte la porzione iniziale della stringa puntata da **nptr** in una rappresentazione **int**. La funzione **atoi** restituirà il valore convertito.

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

Converte la porzione iniziale della stringa puntata da **nptr** in una rappresentazione **long**. La funzione **atol** restituirà il valore convertito.

```
double strtod(const char *nptr, char **endptr);
```

Converte la porzione iniziale della stringa puntata da **nptr** in una rappresentazione **double**. In primo luogo, essa scomporrà la stringa di input in tre parti: una sequenza iniziale, eventualmente vuota, formata da caratteri di spazio bianco (come specificato dalla funzione **isspace**),

una sequenza soggetto somigliante a una costante in virgola mobile e una stringa finale formata da uno o più caratteri non riconosciuti, incluso quello nullo di terminazione della stringa di input. Quindi, essa tenterà di convertire la sequenza soggetto in un numero in virgola mobile e restituirà il risultato.

La forma espansa della sequenza soggetto sarà formata da: un segno positivo o negativo opzionale; una sequenza non vuota di cifre, contenente facoltativamente il carattere separatore dei decimali; una parte esponente opzionale, ma senza suffisso per valori in virgola mobile. La sequenza soggetto è definita come la sottosequenza più lunga della stringa di input, che incominci con il primo carattere diverso da quelli di spazio bianco e che sia della forma attesa. La sequenza soggetto non conterrà dei caratteri, qualora la stringa di input sia vuota, o sia formata interamente da spazi bianchi, o qualora il primo carattere diverso da uno spazio bianco non corrisponda a un segno, una cifra o al carattere separatore dei decimali.

Qualora la sequenza soggetto abbia la forma attesa, la sequenza di caratteri che incomincia con la prima cifra o con il carattere separatore dei decimali (ciò che appare prima) sarà interpretata come una costante in virgola mobile, eccetto che il carattere separatore dei decimali sarà utilizzato in sostituzione di un punto e che, qualora non compaia né un esponente né un carattere separatore dei decimali, si presumerà che questo segua l'ultima cifra della stringa. Qualora la sequenza soggetto cominci con un segno negativo, il valore risultante dalla conversione sarà invertito di segno. Un puntatore alla stringa finale sarà immagazzinato nell'oggetto puntato da **endptr**, a patto che questo non sia un puntatore nullo.

Qualora la sequenza soggetto sia vuota o non abbia la forma attesa, non sarà eseguita alcuna conversione; il valore di **nptr** sarà immagazzinato nell'oggetto puntato da **endptr**, a patto che questo non sia un puntatore nullo.

La funzione **strtod** restituirà il valore convertito, qualora ce ne sia uno. Qualora non possa essere eseguita nessuna conversione, sarà restituito uno zero. Qualora il valore corretto sia esterno all'intervallo di quelli rappresentabili, sarà restituito **HUGE_VAL** positivo o negativo (in accordo con il segno del valore), e sarà immagazzinato in **errno** il valore della macro **ERANGE**. Qualora il valore corretto sia troppo piccolo per essere rappresentato, sarà restituito zero e il valore della macro **ERANGE** sarà immagazzinato in **errno**.

```
long int strtol(const char *nptr, char **endptr, int base);
```

Converte la porzione iniziale della stringa puntata da **nptr** in una rappresentazione **long int**. In primo luogo essa scomporrà la stringa di input in tre parti: una sequenza iniziale, eventualmente vuota, formata da caratteri di spazio bianco (così come specificato dalla funzione **isspace**), una sequenza soggetto somigliante a un intero rappresentato in una qualche base determinata dal valore di **base** e una stringa finale formata da uno o più caratteri non riconosciuti, incluso quello nullo di terminazione della stringa di input. Quindi, essa tenterà di convertire la sequenza soggetto in un intero e restituirà il risultato.

Qualora il valore di **base** sia zero, la forma attesa della sequenza soggetto sarà quella di una costante intera, facoltativamente preceduta da un segno positivo o negativo, ma senza suffisso per gli interi. Qualora il valore di **base** sia compreso tra 2 e 36, la forma attesa sarà una sequenza di lettere e cifre che rappresenterà un intero nella base specificata da **base**, facoltativamente preceduta da un segno positivo o negativo, ma senza suffisso per gli interi. Le lettere da **a** (o **A**) a **z** (o **Z**) saranno associate ai valori da 10 a 35; saranno ammesse soltanto le lettere i cui valori ascritti siano inferiori a quello di **base**. Qualora il

valore di **base** sia 16, i caratteri **0x** o **0X** potranno precedere facoltativamente la sequenza di lettere e cifre e seguire il segno, nel caso sia presente.

La sequenza soggetto è definita come la sottosequenza iniziale più lunga della stringa di input, che incominci con il primo carattere diverso da quelli di spazio bianco e che sia della forma attesa. La sequenza soggetto non conterrà dei caratteri, qualora la stringa di input sia vuota, o sia formata interamente da spazi bianchi, o qualora il primo carattere diverso da uno spazio bianco non corrisponda a un segno, a una lettera o a una cifra ammissibile.

Qualora la sequenza soggetto abbia la forma attesa e il valore di **base** sia zero, la sequenza di caratteri che incomincia con la prima cifra sarà interpretata come una costante intera. Qualora la sequenza soggetto abbia la forma attesa e **base** sia compresa tra 2 e 36, questa sarà utilizzata come base per la conversione, attribuendo a ogni lettera il suo valore così come detto in precedenza. Qualora la sequenza soggetto incominci con un segno negativo, il valore risultante dalla conversione sarà invertito di segno. Un puntatore alla stringa finale sarà immagazzinato nell'oggetto puntato da **endptr**, a patto che questo non sia un puntatore nullo.

Qualora la sequenza soggetto sia vuota o non abbia la forma attesa, non sarà eseguita alcuna conversione; il valore di **nptr** sarà immagazzinato nell'oggetto puntato da **endptr**, a patto che questo non sia un puntatore nullo.

La funzione **strtoul** restituirà il valore convertito, qualora ce ne sia uno. Qualora non possa essere eseguita nessuna conversione, sarà restituito uno zero. Qualora il valore corretto sia esterno all'intervallo di quelli rappresentabili, sarà restituito **LONG_MAX** o **LONG_MIN** (in accordo con il segno del valore) e sarà immagazzinato in **errno** il valore della macro **ERANGE**.

```
unsigned long int strtoul(const char *nptr, char **endptr, int base);
```

Converte la porzione iniziale della stringa puntata da **nptr** in una rappresentazione **unsigned long int**. La funzione **strtoul** funziona in modo identico a **strtoul**. La funzione **strtoul** restituirà il valore convertito, qualora ce ne sia uno. Qualora non possa essere eseguita nessuna conversione, sarà restituito uno zero. Qualora il valore corretto sia esterno all'intervallo di quelli rappresentabili, sarà restituito **ULONG_MAX** e il valore della macro **ERANGE** sarà immagazzinato in **errno**.

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

La funzione **rand** calcola una sequenza di interi pseudocasuali compresi nell'intervallo da 0 a **RAND_MAX**. La funzione **rand** restituirà un intero pseudocasuale.

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

Utilizza l'argomento come seme per una nuova sequenza di numeri pseudocasuali da restituire con le chiamate successive di **rand**. Qualora **srand** sia invocata successivamente con lo stesso valore di seme, la sequenza di numeri pseudocasuali sarà ripetuta. Qualora **rand** sia richiamata prima che sia stata effettuata una qualsiasi invocazione di **srand**, dovrà essere generata la stessa sequenza prodotta quando **srand** viene chiamata per la prima volta con un valore di seme uguale a 1. Le seguenti funzioni definiscono un'implementazione portabile di **rand** e **srand**.

```

static unsigned long int next = 1;

int rand(void) /* si assume che RAND_MAX sia 32767 */
{
    next = next * 1103515245 + 12345;
    return (unsigned int) (next/65536) % 32768;
}

void srand(unsigned int seed)
{
    next = seed;
}

void *calloc(size_t nmemb, size_t dimensione);

```

Alloca uno spazio per un vettore di **nmemb** oggetti, di dimensioni pari a **dimensione**. Lo spazio allocato sarà inizializzato con tutti i bit a zero. La funzione **calloc** restituirà un puntatore nullo o uno che faccia riferimento allo spazio allocato.

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

Fa in modo che lo spazio puntato da **ptr** sia rilasciato, ovvero sia reso disponibile per ulteriori allocazioni. Qualora **ptr** sia un puntatore nullo, non sarà eseguita nessuna azione. In caso contrario, qualora l'argomento non corrisponda a un puntatore restituito in precedenza dalle funzioni **calloc**, **malloc** o **realloc**, o qualora lo spazio sia stato rilasciato da una chiamata a **free** o a **realloc**, il comportamento sarà indefinito.

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

Alloca uno spazio per un oggetto la cui dimensione sarà specificata da **dimensione** e il cui valore sarà indeterminato. La funzione **malloc** restituirà un puntatore nullo o uno che faccia riferimento allo spazio allocato.

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

Cambia la dimensione dell'oggetto puntato da **ptr** con quella specificata da **dimensione**. I contenuti dell'oggetto non saranno modificati entro la dimensione più piccola tra la nuova e la vecchia. Qualora la nuova dimensione sia più grande, il valore della nuova porzione allocata per l'oggetto sarà indeterminato. Qualora **ptr** sia un puntatore nullo, la funzione **realloc** si comporterà come **malloc** per la dimensione specificata. Altrimenti, qualora **ptr** non corrisponda a un puntatore restituito in precedenza dalle funzioni **calloc**, **malloc** o **realloc**, o qualora lo spazio sia stato rilasciato da una chiamata alle funzioni **free** o **realloc**, il comportamento sarà indefinito. Qualora lo spazio non possa essere allocato, l'oggetto puntato da **ptr** resterà immutato. Qualora **dimensione** sia zero e **ptr** non sia nullo, l'oggetto puntato sarà rilasciato. La funzione **realloc** restituirà un puntatore nullo o uno che faccia riferimento allo spazio allocato ed eventualmente spostato.

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

Fa in modo che si verifichi la chiusura anormale del programma, sempre che il segnale **SIGABRT** non sia intercettato e che il relativo gestore non restituisca il controllo. Lo svuotamento degli stream di output aperti, la chiusura degli stream aperti e la rimozione dei file temporanei, dipenderanno dall'implementazione. All'ambiente di esecuzione sarà restituita una

forma definita dall'implementazione dello stato terminazione senza successo, attraverso la chiamata della funzione **raise(SIGABRT)**. La funzione **abort** non potrà restituire il controllo al chiamante.

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

Registra la funzione puntata da **funz** perché sia richiamata senza argomenti alla terminazione normale del programma. L'implementazione dovrà supportare la registrazione di almeno 32 funzioni. La funzione **atexit** restituirà zero qualora la registrazione abbia successo, un valore diverso da zero qualora fallisca.

```
void exit(int stato);
```

Fa in modo che si verifichi la chiusura normale del programma. Qualora un programma esegua più di una chiamata alla funzione **exit**, il comportamento sarà indefinito. In primo luogo, saranno richiamate in ordine inverso alla loro registrazione tutte le funzioni registrate da **atexit**. Ognuna di esse sarà richiamata una volta per ogni registrazione effettuata. In seguito, saranno svuotati tutti gli stream aperti che abbiano nei buffer dei dati non ancora scritti, saranno chiusi tutti gli stream aperti e saranno rimossi tutti i file creati con la funzione **tmpfile**.

Infine, il controllo sarà restituito all'ambiente di esecuzione. Qualora il valore di **stato** sia zero o **EXIT_SUCCESS**, sarà restituita una forma definita dall'implementazione dello stato terminazione con successo. Qualora il valore di **stato** sia **EXIT_FAILURE**, sarà restituita una forma definita dall'implementazione dello stato terminazione senza successo. Altrimenti lo stato restituito sarà definito dall'implementazione. La funzione **exit** non potrà restituire il controllo al suo chiamante.

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

Ricerca all'interno di un elenco di ambiente fornito da quello di esecuzione una stringa corrispondente a quella puntata da **nome**. L'insieme dei nomi dell'ambiente e dei metodi per alterare il relativo elenco saranno definiti dall'implementazione. Restituirà un puntatore a una stringa associata al membro corrispondente dell'elenco. La stringa puntata non dovrebbe essere modificata dal programma, ma potrà essere sostituita da una successiva chiamata alla funzione **getenv**. Qualora il **nome** specificato non possa essere ritrovato, sarà restituito un puntatore nullo.

```
int system(const char *stringa);
```

Passa la stringa puntata da **stringa** all'ambiente di esecuzione perché sia eseguita da un interprete di comandi in un modo definito dall'implementazione. Per **stringa** potrà essere utilizzato un puntatore nullo per verificare se esista un interprete di comandi. Qualora l'argomento sia un puntatore nullo, la funzione **system** restituirà un valore diverso da zero soltanto qualora sia disponibile un interprete di comandi. Qualora l'argomento non sia un puntatore nullo, la funzione **system** restituirà un valore definito dall'implementazione.

```
void *bsearch(const void *chiave, const void *base, size_t nmemb,
              size_t dimensione, int (*compar)(const void *, const void *));
```

Ricerca in un vettore di **nmemb** oggetti, il cui primo elemento sarà puntato da **base**, un oggetto che corrisponda a quello puntato da **chiave**. La dimensione di ogni elemento del vettore sarà specificata da **dimensione**. La funzione di comparazione puntata da **compar**

sarà richiamata con due argomenti che punteranno rispettivamente all'oggetto **chiave** e a un elemento del vettore. La funzione dovrà restituire un intero minore, uguale o maggiore a zero qualora l'oggetto **chiave** sia considerato rispettivamente minore, uguale o maggiore di quello del vettore. Questo dovrà essere formato, nell'ordine, da tutti gli elementi considerati minori, uguali e maggiori dell'oggetto **chiave**.

La funzione **bsearch** restituirà un puntatore all'elemento corrispondente del vettore, o uno nullo qualora non sia stata ritrovata alcuna corrispondenza. Qualora siano considerati uguali due elementi, non sarà specificato quale dei due sarà quello corrispondente.

```
void qsort(void *base, size_t nmemb, size_t dimensione, int
          (*compar)(const void *, const void *));
```

Ordina un vettore di **nmemb** oggetti. L'elemento iniziale sarà puntato da **base**. La dimensione di ogni oggetto sarà specificata da **dimensione**. I contenuti del vettore saranno ordinati in modo ascendente secondo la funzione di comparazione puntata da **compar**; questa sarà richiamata con due argomenti che punteranno agli oggetti che dovranno essere confrontati. La funzione restituirà un intero minore, uguale o maggiore a zero qualora il primo argomento sia considerato rispettivamente minore, uguale o maggiore del secondo. Qualora due elementi siano considerati uguali, il loro ordine all'interno del vettore ordinato non sarà definito.

```
int abs(int j);
```

Calcola il valore assoluto di un intero **j**. Qualora il risultato non possa essere rappresentato, il comportamento sarà indefinito. La funzione **abs** restituirà il valore assoluto.

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

Calcola il quoziente e il resto della divisione del numeratore **numer** per il denominatore **denom**. Qualora la divisione non sia esatta, il quoziente risultante sarà l'intero più vicino per difetto al quoziente algebrico. Qualora il risultato non possa essere rappresentato, il comportamento sarà indefinito; altrimenti, **quoz * denom + res** dovrà essere uguale a **numer**. La funzione **div** restituirà una struttura di tipo **div_t**, comprendente il quoziente e il resto. La struttura dovrà contenere i seguenti membri, in qualsiasi ordine:

```
int quoz; /* quoziente */
int res; /* resto */
long int labs(long int j);
```

Simile alla funzione **abs**, eccetto che l'argomento e il valore restituito saranno di tipo **long int**.

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

Simile alla funzione **div**, eccetto che gli argomenti e i membri della struttura restituita (che sarà di tipo **ldiv_t**) saranno tutti dei **long int**.

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

Qualora **s** non sia un puntatore nullo, la funzione **mblen** determinerà il numero dei byte contenuti nel carattere multibyte puntato da **s**. Qualora **s** sia un puntatore nullo, la funzione **mblen** restituirà un valore diverso da zero o zero, a seconda che la codifica del carattere multibyte sia o no dipendente dallo stato. Qualora **s** non sia un puntatore nullo, la funzione **mblen**

restituirà 0 (nel caso che **s** punti al carattere nullo), oppure il numero di byte del carattere multibyte (nel caso che i successivi **n** o meno caratteri formino un multibyte valido), oppure -1 (nel caso che i suddetti non formino un carattere multibyte valido).

```
int mbtowl(wchar_t *pwc, const char *s, size_t n);
```

Qualora **s** non sia un puntatore nullo, la funzione **mbtowl** determinerà il numero dei byte contenuti nel carattere multibyte puntato da **s**. Essa quindi determinerà il codice per il valore di tipo **wchar_t** che corrisponda a quel carattere multibyte. (Il valore del codice corrispondente al carattere nullo è zero). Qualora il carattere multibyte sia valido e **pwc** non sia un puntatore nullo, la funzione **mbtowl** immagazzinerà il codice nell'oggetto puntato da **pwc**. Saranno esaminati al massimo **n** byte del vettore puntato da **s**.

Qualora **s** sia un puntatore nullo, la funzione **mbtowl** restituirà un valore diverso da zero o zero, a seconda che la codifica del carattere multibyte sia o no dipendente dallo stato. Qualora **s** non sia un puntatore nullo, la funzione **mbtowl** restituirà 0 (nel caso che **s** punti al carattere nullo), oppure il numero di byte contenuti nel carattere multibyte convertito (nel caso che i successivi **n** o meno byte formino un multibyte valido), oppure -1 (nel caso che i suddetti non formino un carattere multibyte valido). In nessun caso il valore restituito sarà maggiore di **n** o di quello della macro **MB_CUR_MAX**.

```
int wctomb(char *s, wchar_t wchar);
```

La funzione **wctomb** determina il numero di byte necessario per rappresentare il carattere multibyte corrispondente al codice il cui valore sarà contenuto in **wchar** (inclusa ogni modifica allo stato relativo al tasto delle maiuscole). Essa immagazzinerà la rappresentazione del carattere multibyte nell'oggetto di tipo vettore puntato da **s** (qualora **s** non sia un puntatore nullo). Saranno immagazzinati un massimo di **MB_CUR_MAX** caratteri. Qualora il valore di **wchar** sia zero, la funzione **wctomb** rimarrà nello stato iniziale relativo al tasto delle maiuscole.

Qualora **s** sia un puntatore nullo, la funzione **wctomb** restituirà un valore diverso da zero o zero, a seconda che la codifica del carattere multibyte sia o no dipendente dallo stato. Qualora **s** non sia un puntatore nullo, la funzione **wctomb** restituirà -1 qualora il valore di **wchar** non corrisponda a un carattere multibyte valido, oppure restituirà il numero di byte del carattere multibyte corrispondente al valore di **wchar**. In nessun caso il valore restituito sarà maggiore di quello della macro **MB_CUR_MAX**.

```
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
```

La funzione **mbstowcs** legge una sequenza di caratteri multibyte, che incominciano nello stato iniziale relativo al tasto delle maiuscole, dal vettore puntato da **s** e la converte nella sequenza di codici corrispondenti, immagazzinandone un massimo di **n** nel vettore puntato da **pwcs**. Non sarà esaminato o convertito nessun carattere multibyte successivo a quello nullo (che sarà convertito in un codice di valore zero). Ogni carattere multibyte sarà convertito come lo farebbe un'invocazione della funzione **mbtowl**, eccetto che lo stato relativo al tasto delle maiuscole nella funzione **mbtowl** non sarà influenzato.

Nel vettore puntato da **pwcs** saranno modificati al massimo **n** elementi. Il comportamento di una copia tra oggetti che si sovrappongano sarà indefinito. Qualora sia incontrato un carattere multibyte non valido, la funzione **mbstowcs** restituirà (**size_t**) - 1. Altrimenti,

la funzione **mbstowcs** restituirà il numero degli elementi modificati nel vettore, senza includere il codice zero di terminazione, qualora ce ne sia uno.

```
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
```

La funzione **wcstombs** legge dal vettore puntato da **pwcs** una sequenza di codici che corrispondono a dei caratteri multibyte, convertendola in una sequenza di caratteri multibyte che incominciano nello stato iniziale relativo al tasto delle maiuscole, e immagazzina i caratteri multibyte nel vettore puntato da **s**, fermandosi qualora un carattere multibyte faccia superare il limite di **n** byte, o qualora sia stato immagazzinato un carattere nullo. Ogni codice sarà convertito come lo farebbe un'invocazione della funzione **wctomb**, eccetto che in questo caso lo stato relativo al tasto delle maiuscole nella funzione **wctomb** non sarà influenzato.

Nel vettore puntato da **s** saranno modificati al massimo **n** byte. Qualora la copia avvenga tra oggetti che si sovrappongono, il risultato sarà indefinito. Qualora sia incontrato un codice che non corrisponda a un carattere multibyte valido, la funzione **wcstombs** restituirà **(size_t) - 1**. Altrimenti, la funzione **wcstombs** restituirà il numero dei byte modificati, senza includere il carattere nullo di terminazione, qualora ce ne sia uno.

G.12 Gestione delle stringhe <string.h>

NULL

Una costante di tipo puntatore nullo definita dall'implementazione.

```
size_t
```

Il tipo intero senza segno restituito dall'operatore **sizeof**.

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

La funzione **memcpy** copia **n** caratteri dall'oggetto puntato da **s2** in quello puntato da **s1**. Qualora la copia avvenga tra oggetti che si sovrappongono, il comportamento sarà indefinito. La funzione **memcpy** restituirà il valore di **s1**.

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

La funzione **memmove** copia **n** caratteri dall'oggetto puntato da **s2** in quello puntato da **s1**. La copia avverrà come se i caratteri dell'oggetto puntato da **s2** fossero prima copiati in un vettore temporaneo di **n** caratteri, che non si sovrapponga agli oggetti puntati da **s1** e **s2**, e quindi fossero ricopiati dal vettore temporaneo nell'oggetto puntato da **s1**. La funzione **memmove** restituirà il valore di **s1**.

```
char *strcpy(char *s1, const char *s2);
```

La funzione **strcpy** copia la stringa puntata da **s2** (incluso il carattere nullo di terminazione) nel vettore puntato da **s1**. Qualora la copia avvenga tra oggetti che si sovrappongono, il comportamento sarà indefinito. La funzione **strcpy** restituirà il valore di **s1**.

```
char *strncpy(char *s1, const char *s2, size_t n);
```

La funzione **strncpy** copia un massimo di **n** caratteri, escludendo quelli successivi a quello nullo, dal vettore puntato da **s2** in quello puntato da **s1**. Qualora la copia avvenga tra

oggetti che si sovrappongano, il comportamento sarà indefinito. Qualora il vettore puntato da **s2** sia una stringa più corta di **n** caratteri, nella copia del vettore puntato da **s1** saranno accodati dei caratteri nulli, finché non sarà stato raggiunto il numero indicato da **n**. La funzione **strncpy** restituirà il valore di **s1**.

```
char *strcat(char *s1, const char *s2);
```

La funzione **strcat** accoda una copia della stringa puntata da **s2** (incluso il carattere nullo di terminazione) alla fine di quella puntata da **s1**. Il carattere iniziale di **s2** si sostituirà a quello di terminazione di **s1**. Qualora la copia avvenga tra oggetti che si sovrappongano, il comportamento sarà indefinito. La funzione **strcat** restituirà il valore di **s1**.

```
char *strncat(char *s1, const char *s2, size_t n);
```

La funzione **strncat** accoda un massimo di **n** caratteri (quello nullo e quelli che lo seguono non saranno accodati) dal vettore puntato da **s2** alla fine della stringa indicata da **s1**. Il carattere iniziale di **s2** si sostituirà a quello nullo di terminazione di **s1**. Al risultato sarà sempre accodato un carattere nullo di terminazione. Qualora la copia avvenga tra oggetti che si sovrappongano, il comportamento sarà indefinito. La funzione **strncat** restituirà il valore di **s1**.

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

La funzione **memcmp** confronta i primi **n** caratteri dell'oggetto puntato da **s1** con i primi **n** caratteri di quello puntato da **s2**. La funzione **memcmp** restituirà un intero maggiore, uguale o minore di zero qualora l'oggetto puntato da **s1** sia maggiore, uguale o minore di quello puntato da **s2**.

```
int strcmp(const char *s1, const char *s2);
```

La funzione **strcmp** confronta la stringa puntata da **s1** con quella puntata da **s2**. La funzione **strcmp** restituirà un intero maggiore, uguale o minore di zero qualora l'oggetto puntato da **s1** sia maggiore, uguale o minore di quello puntato da **s2**.

```
int strcoll(const char *s1, const char *s2);
```

La funzione **strcoll** confronta la stringa puntata da **s1** con quella puntata da **s2**, interpretando entrambe in modo conforme a quanto indicato dalla categoria **LC_COLLATE** della localizzazione corrente. La funzione **strcoll** restituirà un intero maggiore, uguale o minore di zero qualora la stringa puntata da **s1** sia maggiore, uguale o minore di quella puntata da **s2**, quando entrambe siano interpretate in modo conforme alla localizzazione corrente.

```
int strncmp(const char *s1, const char *s2, size_t n);
```

La funzione **strncmp** confronta un massimo di **n** caratteri del vettore puntato da **s1**, escludendo quelli successivi a quello nullo, con i caratteri corrispondenti del vettore puntato da **s2**. La funzione **strncmp** restituirà un valore maggiore, uguale o minore di zero qualora il vettore puntato da **s1**, eventualmente terminato dal carattere nullo, sia maggiore, uguale o minore di quello puntato da **s2**, eventualmente terminato dal carattere nullo.

```
size_t strxfrm(char *s1, const char *s2, size_t n);
```

La funzione **strxfrm** trasforma la stringa puntata da **s2**, inserendo quella risultante nel vettore indicato da **s1**. La trasformazione sarà tale che, qualora la funzione **strcmp** fosse

applicata a due stringhe trasformate, essa restituirebbe un valore maggiore, uguale o minore di zero corrispondente al risultato della funzione **strcoll** applicata alle stesse due stringhe originali. Nel vettore risultante puntato da **s1** saranno inseriti al massimo **n** caratteri, incluso quello nullo di terminazione. Qualora **n** sia zero, **s1** potrà essere un puntatore nullo. Qualora la copia avvenga tra oggetti che si sovrappongono, il comportamento sarà indefinito. La funzione **strxfrm** restituirà la lunghezza della stringa trasformata (escluso il carattere nullo di terminazione). Qualora il valore restituito sia pari o maggiore di **n**, i contenuti del vettore puntato da **s1** saranno indeterminati.

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

La funzione **memchr** individua la prima occorrenza di **c** (convertito in un **unsigned char**) nei primi **n** caratteri (ognuno interpretato come **unsigned char**) dell'oggetto puntato da **s**. La funzione **memchr** restituirà un puntatore al carattere individuato, o uno nullo qualora il carattere non sia presente nell'oggetto.

```
char *strchr(const char *s, int c);
```

La funzione **strchr** individua la prima occorrenza di **c** (convertito in un **char**) nella stringa puntata da **s**. Il carattere nullo di terminazione sarà considerato parte della stringa. La funzione **strchr** restituirà un puntatore al carattere individuato, o uno nullo qualora il carattere non sia presente nella stringa.

```
size_t strcspn(const char *s1, const char *s2);
```

La funzione **strcspn** calcola la lunghezza del segmento iniziale massimo che, nella stringa puntata da **s1**, sia formato interamente da caratteri non contenuti nella stringa puntata da **s2**. La funzione **strcspn** restituirà la lunghezza del segmento.

```
char *strpbrk(const char *s1, const char *s2);
```

La funzione **strpbrk** individua la prima occorrenza nella stringa puntata da **s1** di qualsiasi carattere incluso in quella puntata da **s2**. La funzione **strpbrk** restituirà un puntatore al carattere, o uno nullo qualora nessuno di quelli inclusi in **s2** sia presente in **s1**.

```
char *strrchr(const char *s, int c);
```

La funzione **strrchr** individua l'ultima occorrenza di **c** (convertito in un **char**) nella stringa puntata da **s**. Il carattere nullo di terminazione sarà considerato parte della stringa. La funzione **strrchr** restituirà un puntatore al carattere, o uno nullo qualora **c** non sia presente nella stringa.

```
size_t strspn(const char *s1, const char *s2);
```

La funzione **strspn** calcola la lunghezza del segmento iniziale massimo che, nella stringa puntata da **s1**, sia formato interamente da caratteri contenuti nella stringa puntata da **s2**. La funzione **strspn** restituirà la lunghezza del segmento.

```
char *strstr(const char *s1, const char *s2);
```

La funzione **strstr** individua, nella stringa puntata da **s1**, la prima occorrenza della sequenza di caratteri (escluso quello nullo di terminazione) inclusa nella stringa puntata da **s2**. La funzione **strstr** restituirà un puntatore alla stringa individuata, o uno nullo qualo-

ra non sia stata ritrovata nessuna corrispondenza. Qualora **s2** punti a una stringa di lunghezza zero, la funzione restituirà **s1**.

```
char *strtok(char *s1, const char *s2);
```

Una serie di invocazioni della funzione **strtok** suddividerà la stringa puntata da **s1** in una sequenza di token, ognuno dei quali sarà delimitato da un carattere tra quelli inclusi nella stringa puntata da **s2**. La prima invocazione della serie avrà **s1** come suo argomento e sarà seguita da invocazioni che abbiano un puntatore nullo come loro primo argomento. La stringa dei separatori puntata da **s2** potrà cambiare da una chiamata all'altra.

La prima invocazione della serie ricercherà, nella stringa puntata da **s1**, il primo carattere che non sia contenuto in quella dei separatori puntata da **s2**. Qualora il suddetto carattere non sia ritrovato, allora non ci saranno token nella stringa puntata da **s1** e la funzione **strtok** restituirà un puntatore nullo. Qualora il suddetto carattere sia stato ritrovato, questo corrisponderà all'inizio del primo token.

La funzione **strtok** ricercherà quindi da quel punto un carattere che sia tra quelli contenuti nella stringa dei separatori. Qualora il suddetto carattere non sia stato ritrovato, il token corrente si estenderà fino alla fine della stringa puntata da **s1** e le successive ricerche di un token restituiranno un puntatore nullo. Qualora il suddetto carattere sia stato ritrovato, questo sarà sostituito da uno nullo che terminerà il token corrente. La funzione **strtok** salverà un puntatore al carattere successivo, dal quale partirà la prossima ricerca di un token.

Ogni chiamata successiva, che abbia un puntatore nullo come valore del primo argomento, inizierà la ricerca dal punto salvato e si comporterà come descritto prima. L'implementazione dovrà comportarsi come se nessuna funzione della libreria richiami **strtok**. La funzione **strtok** restituirà un puntatore al primo carattere di un token, o uno nullo qualora non ce ne siano.

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

La funzione **memset** copia il valore di **c**, convertito in un **unsigned char**, in ognuno dei primi **n** caratteri dell'oggetto puntato da **s**. La funzione **memset** restituirà il valore di **s**.

```
char *strerror(int errnum);
```

La funzione **strerror** individua la stringa del messaggio di errore corrispondente al valore di **errnum**. L'implementazione dovrà comportarsi come se nessuna funzione della libreria richiami **strerror**. La funzione **strerror** restituirà un puntatore alla stringa, i cui contenuti saranno definiti dall'implementazione. Il vettore puntato non dovrebbe essere modificato dal programma, ma potrà essere sostituito da una chiamata successiva alla funzione **strerror**.

```
size_t strlen(const char *s);
```

La funzione **strlen** calcola la lunghezza della stringa puntata da **s**. La funzione **strlen** restituirà il numero di caratteri che precedono quello nullo di terminazione.

G.13 Data e ora <time.h>

CLOCKS_PER_SEC

Il numero per secondo del valore restituito dalla funzione **clock**.

NULL

Una costante di tipo puntatore nullo definita dall'implementazione.

`clock_t`

Un tipo aritmetico in grado di rappresentare l'ora.

`time_t`

Un tipo aritmetico in grado di rappresentare l'ora.

`size_t`

Il tipo intero senza segno restituito dell'operatore **sizeof**.

`struct tm`

Mantiene i componenti delle date, chiamati tempo frammentato. La struttura dovrà contenere almeno i seguenti membri, in qualsiasi ordine. Le semantiche dei membri e dei loro normali intervalli sono espresse nei commenti.

```
int tm_sec;      /* secondi dopo il minuto: [0, 59] */
int tm_min;     /* minuti dopo l'ora: [0, 59] */
int tm_hour;    /* ore dalla mezzanotte: [0, 23] */
int tm_mday;   /* giorno del mese: [1, 31] */
int tm_mon;    /* mese da gennaio: [0, 11] */
int tm_year;   /* anno dal 1900 */
int tm_wday;   /* giorni dalla domenica: [0, 6] */
int tm_yday;   /* giorni dal 1° gennaio: [0, 365] */
int tm_isdst;  /* flag per l'ora legale */
```

Il valore di **tm_isdst** sarà positivo qualora l'ora legale sia in corso, zero qualora non lo sia e negativo qualora l'informazione non sia disponibile.

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

La funzione **clock** determina il tempo di processore utilizzato. La funzione **clock** restituirà la miglior approssimazione dell'implementazione per quanto riguarda il tempo del processore usato dal programma, dall'inizio di un momento definito dall'implementazione e correlato soltanto con l'invocazione del programma. Per determinare il tempo in secondi, il valore restituito dalla funzione **clock** dovrà essere diviso per quello della macro **CLOCKS_PER_SEC**. Qualora il tempo usato del processore non sia disponibile o il suo valore non possa essere rappresentato, la funzione restituirà il valore **(clock_t) - 1**.

```
double difftime(time_t time1, time_t time0);
```

La funzione **difftime** calcola la differenza tra due date: **time1 - time0**. La funzione **difftime** restituirà la differenza espressa in secondi in un **double**.

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

La funzione **mktime** converte il tempo frammentato, espresso come ora locale, della struttura puntata da **timeptr** in un valore di tipo data con la stessa codifica di quelli restituiti dalla funzione **time**. I valori originali dei membri **tm_wday** e **tm_yday** della

struttura saranno ignorati, mentre quelli degli altri membri non saranno limitati agli intervalli indicati prima. In un completamento successivo, i valori dei membri **tm_wday** e **tm_yday** della struttura saranno impostati in modo appropriato, mentre gli altri membri saranno impostati in modo da rappresentare la data specificata, ma con i loro valori forzati negli intervalli indicati prima; il valore finale di **tm_mday** non sarà impostato finché **tm_mon** e **tm_year** non saranno stati determinati. La funzione **mktime** restituirà la data specificata codificata come un valore di tipo **time_t**. Qualora la data non possa essere rappresentata, la funzione restituirà il valore **(time_t) - 1**.

```
time_t time(time_t *timer);
```

La funzione **time** determina la data e l'ora corrente. Essa restituisce la miglior approssimazione dell'implementazione per quanto riguarda la data e l'ora correnti. Il valore **(time_t) - 1** sarà restituito qualora la data e l'ora non siano disponibili. Qualora **timer** non sia un puntatore nullo, il valore restituito sarà assegnato anche all'oggetto puntato dallo stesso.

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

La funzione **asctime** converte il tempo frammentato della struttura puntata da **timeptr** in una stringa dal seguente formato

```
Sun Sep 16 01:03:52 1973\n\0
```

La funzione **asctime** restituirà un puntatore alla stringa.

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

La funzione **ctime** converte la data e l'ora puntate da **timer** in una stringa che rappresenti l'ora locale. È equivalente a

```
asctime(localtime(timer))
```

La funzione **ctime** restituirà il puntatore ricevuto dalla chiamata della funzione **asctime** con quell'argomento di tempo frammentato.

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

La funzione **gmtime** converte la data e l'ora puntate da **timer** in un tempo frammentato, espresso come Coordinated Universal Time (UTC, ora mondiale coordinata). La funzione **gmtime** restituirà un puntatore a quell'oggetto, o uno nullo qualora l'UTC non sia disponibile.

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

La funzione **localtime** converte la data e l'ora puntate da **timer** in un tempo frammentato espresso come ora locale. La funzione **localtime** restituirà un puntatore a quell'oggetto.

```
size_t strftime(char *s, size_t maxdim, const char *formato, const
                struct tm *timeptr);
```

La funzione **strftime** inserisce dei caratteri nel vettore puntato da **s** nel modo indicato dalla stringa puntata da **formato**. Questa sarà formata da zero o più specifiche di conversione e da caratteri multibyte ordinari. Tutti i caratteri ordinari (incluso quello nullo di terminazione) saranno copiati senza modifiche nel vettore. Qualora la copia avvenga tra oggetti che si sovrappongono, il comportamento sarà indefinito. Nel vettore saranno inseriti al massimo **maxdim** caratteri. Ogni specifica di conversione sarà sostituita dai caratteri appropriati come

descritto nella lista seguente. I caratteri appropriati saranno determinati dalla categoria **LC_TIME** della localizzazione corrente e dai valori contenuti nella struttura puntata da **timeptr**.

- %a** sarà sostituita dal nome locale abbreviato del giorno della settimana.
- %A** sarà sostituita dal nome locale completo del giorno della settimana.
- %b** sarà sostituita dal nome locale abbreviato del mese.
- %B** sarà sostituita dal nome locale completo del mese.
- %c** sarà sostituita dall'appropriata rappresentazione locale della data e dell'ora.
- %d** sarà sostituita dal giorno del mese espresso come numero decimale (**01-31**).
- %H** sarà sostituita dall'ora (nel formato di 24 ore) espressa come numero decimale (**00-23**).
- %I** sarà sostituita dall'ora (nel formato di 12 ore) espressa come numero decimale (**01-12**).
- %j** sarà sostituita dal giorno dell'anno espresso come numero decimale (**001-366**).
- %m** sarà sostituita dal mese espresso come numero decimale (**01-12**).
- %M** sarà sostituita dal minuto espresso come numero decimale (**00-59**).
- %p** sarà sostituita dall'equivalente locale della designazione AM/PM associata ad un formato di 12 ore.
- %S** sarà sostituita dal secondo espresso come numero decimale (**00-59**).
- %U** sarà sostituita dal numero della settimana dell'anno (la prima domenica sarà il primo giorno della settimana 1) espresso come numero decimale (**00-53**).
- %w** sarà sostituita dal giorno della settimana espresso come numero decimale (**0-6**), dove la domenica sarà **0**.
- %W** sarà sostituita dal numero della settimana dell'anno (il primo lunedì sarà il primo giorno della settimana 1) espresso come numero decimale (**00-53**).
- %x** sarà sostituita dall'appropriata rappresentazione locale della data.
- %X** sarà sostituita dall'appropriata rappresentazione locale dell'ora.
- %y** sarà sostituita dall'anno senza il secolo espresso come numero decimale (**00-99**).
- %Y** sarà sostituita dall'anno (compreso il secolo) espresso come numero decimale.
- %Z** sarà sostituita dal nome o dall'abbreviazione della zona del fuso orario, o da nessun carattere qualora la zona del fuso orario non sia determinabile.
- %%** sarà sostituita da **%**.

Nel caso che una specifica di conversione non sia una di quelle elencate sopra, il comportamento sarà indefinito. Qualora il numero totale di caratteri risultanti, incluso quello nullo di terminazione, non sia maggiore di **maxdim**, la funzione **strftime** restituirà il numero di quelli inseriti nel vettore puntato da **s**, senza includere quello nullo di terminazione. Altrimenti, sarà restituito il valore zero e i contenuti del vettore saranno indeterminati.

G.14 Limiti dell'implementazione

<limits.h>

Le seguenti costanti simboliche dovranno essere definite con grandezza (valore assoluto) uguale o maggiore ai valori indicati di seguito.

```
#define CHAR_BIT 8
```

Il numero di bit per l'oggetto più piccolo che non sia un campo di bit (byte).

```
#define SCHAR_MIN -127
```

Il valore minimo per un oggetto di tipo **signed char**.

```
#define SCHAR_MAX +127
```

Il valore massimo per un oggetto di tipo **signed char**.

```
#define UCHAR_MAX 255
```

Il valore massimo per un oggetto di tipo **unsigned char**.

```
#define CHAR_MIN 0 0 SCHAR_MIN
```

Il valore minimo per un oggetto di tipo **char**.

```
#define CHAR_MAX UCHAR_MAX 0 SCHAR_MAX
```

Il valore massimo per un oggetto di tipo **char**.

```
#define MB_LEN_MAX 1
```

Il numero massimo di byte in un carattere multibyte, per ogni localizzazione supportata.

```
#define SHRT_MIN -32767
```

Il valore minimo per un oggetto di tipo **short int**.

```
#define SHRT_MAX +32767
```

Il valore massimo per un oggetto di tipo **short int**.

```
#define USHRT_MAX 65535
```

Il valore massimo per un oggetto di tipo **unsigned short int**.

```
#define INT_MIN -32767
```

Il valore minimo per un oggetto di tipo **int**.

```
#define INT_MAX +32767
```

Il valore massimo per un oggetto di tipo **int**.

```
#define UINT_MAX 65535
```

Il valore massimo per un oggetto di tipo **unsigned int**.

```
#define LONG_MIN -2147483647
```

Il valore minimo per un oggetto di tipo **long int**.

```
#define LONG_MAX +2147483647
```

Il valore massimo per un oggetto di tipo **long int**.

```
#define ULONG_MAX 4294967295
```

Il valore massimo per un oggetto di tipo **unsigned long int**.

<float.h>

```
#define FLT_ROUNDS
```

La modalità di arrotondamento per l'addizione in virgola mobile.

- 1 indeterminabile
- 0 prossima allo zero
- 1 al più vicino
- 2 prossima all'infinito positivo
- 3 prossima all'infinito negativo

Le seguenti costanti simboliche dovranno essere definite con grandezza (valore assoluto) uguale o maggiore dei valori indicati di seguito.

```
#define FLT_RADIX 2
```

La base della rappresentazione dell'esponente, b.

```
#define FLT_MANT_DIG
#define LDBL_MANT_DIG
#define DBL_MANT_DIG
```

Il numero di cifre in base **FLT_RADIX** nel significando in virgola mobile, p.

```
#define FLT_DIG 6
#define DBL_DIG 10
#define LDBL_DIG 10
```

Il numero di cifre decimali, q, tale che ogni numero in virgola mobile con q cifre decimali possa essere arrotondato in un numero in virgola mobile con base p e b cifre e riportato al suo valore originario senza modifiche alle q cifre decimali.

```
#define FLT_MIN_EXP
#define DBL_MIN_EXP
#define LDBL_MIN_EXP
```

L'intero negativo minimo tale che **FLT_RADIX** elevato a quella potenza meno 1 sia un numero in virgola mobile normalizzato.

```
#define FLT_MIN_10_EXP -37
#define DBL_MIN_10_EXP -37
#define LDBL_MIN_10_EXP -37
```

L'intero negativo minimo tale che 10 elevato a quella potenza sia compreso nell'intervallo dei numeri in virgola mobile normalizzati.

```
#define FLT_MAX_EXP
#define DBL_MAX_EXP
#define LDBL_MAX_EXP
```

L'intero massimo tale che **FLT_RADIX** elevato a quella potenza meno 1 sia un numero finito rappresentabile in virgola mobile.

```
#define FLT_MAX_10_EXP          +37
#define DBL_MAX_10_EXP          +37
#define LDBL_MAX_10_EXP        +37
```

L'intero massimo tale che 10 elevato a quella potenza sia compreso nell'intervallo dei numeri finiti rappresentabili in virgola mobile.

Le seguenti costanti simboliche dovranno essere definite uguali o maggiori dei valori mostrati di seguito.

```
#define FLT_MAX                  1E+37
#define DBL_MAX                  1E+37
#define LDBL_MAX                 1E+37
```

Il numero massimo finito in virgola mobile rappresentabile.

Le seguenti costanti simboliche dovranno essere definite uguali o minori dei valori mostrati di seguito.

```
#define FLT_EPSILON              1E-5
#define DBL_EPSILON              1E-9
#define LDBL_EPSILON             1E-9
```

La differenza tra il valore 1,0 e quello minimo maggiore di 1,0 che sia rappresentabile nel tipo in virgola mobile specificato.

```
#define FLT_MIN                  1E-37
#define DBL_MIN                  1E-37
#define LDBL_MIN                 1E-37
```

Il numero positivo minimo normalizzato in virgola mobile.

(Diritti d'autore: questo materiale è stato condensato e adattato a partire dal documento American National Standard for Information Systems—Programming Language—C, ANSI/ISO 9899: 1990. Copie di questo standard possono essere acquistate dalla American National Standards Institute, West 42nd Street, New York, NY 10036.)

