La gestione dell'Errore

Esiste una variabile globale intera, definita nell'header errno.h che viene settata nel caso in cui una chiamata di sistema non possa eseguire correttamente il suo compito. Tale variabile allora indica il tipo di errore avvenuto.

```
#include <errno.h>
extern int errno;
```

Tale variabile può essere letta e scritta come ogni altra variabile.
In particolare il valore di questa variabile serve come indice per una variabile di sistema che è un vettore di stringhe. Queste stringhe contengono dei messaggi di errore caratteristici dell'errore avvenuto.

```
#include <stdio.h>
const char *sys_errlist[];
```

Nel caso una funzione di sistema ci avvisi di un errore possiamo farci visualizzare a video (sullo standard error) la particolare stringa che ci notifica l'errore, utilizzando la funzione

void perror(const char *str);

La stringa puntata da str viene visualizzata prima del messaggio di errore.
Se l'ultima chiamata di sistema non ha causato errore, non si deve usare errno, perchè il suo valore non è definito.

```
#include <stdio.h>
#include <errno.h>
void apri_file(void) {
FILE *finput;
finput=fopen("c:\\users\\input.txt","rt");
if ( finput==NULL ) {
   perror("errore in funzione apri_file: ");
   exit(0);
  }
```

buffers di sistema, fine file ed errori.

Quando un file precedentemente aperto non serve più, deve essere chiuso con la chiamata alla funzione di libreria

int fclose(FILE *f);

che restituisce 0 in caso vada tutto bene, 1 in caso di errore.
Poichè l'I/O con le funzioni viste è bufferizzato, potrebbero essere rimaste ancora delle operazioni di scrittura da completare, cioè qualche byte scritto sul file in precedenza potrebbe essere ancora in un buffer temporaneo, e non essere stato ancora fisicamente scritto sul file. Con la chiamata alla fclose si effettuano definitivamente eventuali operazioni di scrittura rimaste in sospeso.

La funzione

int fflush(FILE *f);

effettua esplicitamente lo svuotamento dei buffer facendo completare le operazioni rimaste in sospeso riguardanti lo stream di output indicato da f.
Serve ad assicurarci che ad un certo istante le operazioni precedenti siano state effettuate. Può servire ad esempio dopo alcune printf per far effettivamente scrivere i caratteri sul video. In qualche caso infatti tale operazione potrebbe essere dilazionata.
fflush() restituisce 0 in caso vada tutto bene, EOF in caso di errore.

La funzione

int feof(FILE *f);

restituisce un valore diverso da 0 se la posizione corrente del file ha raggiunto la fine del file. Restituisce 0 se non siamo alla fine del file.

La funzione

int ferror(FILE *f);

restituisce un valore diverso da 0 se lo stream ha verificato un qualche tipo di errore. Restituisce 0 se non siamo alla fine del file.

INPUT / OUTPUT di BLOCCHI di byte da file

ANSI C mette a disposizione anche delle primitive che permettono di leggere/scrivere su un file un blocco di byte di una dimensione specificata. I prototipi di queste finzioni sono anch'essi contenuti nell'header stdio.h, e tali funzioni sono:

size_t fread(void *ptr, size_t size, size_t number, FILE *finput);

cerca di leggere dal file **finput** un blocco di byte formato da **number** blocchi ciascuno di **size** byte. I dati letti sono scritti nell'area dati puntata da **ptr**, che deve ovviamente essere correttamente allocata, cioè di dimensioni sufficienti.

fread restituisce il numero di blocchi letti, e non il numero di byte letti.

Se all'inizio della lettura si incontra la fine del file la funzione fread restituisce 0, perchè non riesce a leggere neanche un blocco.

Però anche in caso di un qualche tipo di errore fread restituisce 0.

Quindi se fread restituisce zero è necessario utilizzare le funzioni **ferror** o **feof** per capire se è accaduto un errore o se si è raggiunta la fine del file.

size_t fwrite(const void *ptr, size_t size, size_t number, FILE*foutput);

cerca di scrivere sul file **foutput** un blocco di byte formato da **number** blocchi ciascuno di **size** byte, copiandoli dall'area dati puntata da **ptr**.

fread restituisce il numero di blocchi scritti, e non il numero di byte scritti.

Come per la fread in caso di un qualche tipo di errore fwrite() restituisce 0.

Input di tipo diverso: PARAMETRI a RIGA di COMANDO

ANSI C mette a disposizione il modo di passare al programma dei parametri nel momento in cui questo viene lanciato. Partendo ad es. da una shell di comandi di Windows NT (una finestra simil DOS per intenderci) possiamo far seguire al nome del programma una serie di caratteri separati da spazi, che verranno copiati e passati sotto forma di stringhe in un vettore di puntatori a char passati come argomento al main.
Il main dovrà essere a tal scopo definito come nell'esempio seguente, dove:
- argc indica il numero di parametri passati più uno (il nome del programma).
- argv è un vettore di stringhe che contiene la riga di comando, in prima posizione il nome del programma lanciato, nelle seguenti i parametri passati
Il programma stampa il numero dei parametri passati a riga di comando compreso il nome del programma, e poi li stampa ad uno ad uno

```
#include <stdio.h>
int main(int argc, char *argv[])
{
int i;
printf("argc=%d   { ",argc);
for ( i=0; i<argc; i++)
        printf("%d:%s ", i, argv[i]);
printf("}\n");
}
```

Se il programma si chiama param, puo' essere eseguito "lanciandolo" dalla shell di comando mediante una delle seguenti linee di comando, ottenendo come output:

linea di comando	output
param calvin hobbes 3.0 40	argc=5 { 0:param 1:calvin 2:hobbes 3:3.0 4:40}
param	argc=1 { 0:param }
param calvin	argc=2 { 0:param 1:calvin }

I/O di Basso Livello: i descrittori di file

Abbiamo in precedenza visto che è possibile effettuare I/O da file, utilizzando primitive di tipo fopen, fread fwrite, fscanf, fgets. Tali operazioni sono però limitate ai file.
Quello che si vorrebbe avere a disposizione sono delle primitive che possono agire indifferentemente su file o su dispositivi di I/O di tipo diverso, quali interfacce di rete, interfacce seriali ecc. Si vorrebbe cioè poter utilizzare una stessa primitiva per accedere a entità di tipo diverso.
Esistono a tale scopo delle funzioni di librerie cosiddette per **l'I/O di basso livello**, che permettono di operare su diversi device. Vengono ad esempio utilizzate per scrivere applicazioni per comunicazioni via rete.
Ogni dispositivo per I/O, verrà aperto con una funzione detta open che restituisce un intero detto "**descrittore di file**". Questo numero intero rappresenta un indice per una struttura dati che descrive le modalità con cui operare sul dispositivo. Tale descrittore verrà utilizzato in tutte le chiamate per l'effettuazione di I/O.
Poichè queste istruzioni sono di basso livello, non esiste più distinzione tra file di testo e file binari. Tutti i file sono visti come concatenazione di byte.

Un esempio d'uso di queste primitive è qui di seguito riportato.

```
/*leggo i byte di un file binario. Il nome del file e' nella linea di comandi*/
#include <stdio.h>
#include <fcntl.h>
main(int argc,char **argv)
{  int fd;   int bytes_read; char byte;
   if((fd=open(argv[1],O_RDONLY))=-1) exit(1); /*errore,file non aperto*/
   while ( (bytes_read=read(fd,&byte,1))>0 )        /* leggo un byte */
   {
      .... uso il byte letto .....
   }
   if (bytes_read==-1)     exit(1); /* errore in lettura file */
   close(fd);
}
```

#include <stdio.h>
#include <fcntl.h>

Per aprire un file si usa la funzione:

int open(char *filename, int flag, int perms)

che ritorna un file descriptor con valore >=0, oppure -1 se l'operazione fallisce.

Il parametro **flag** controlla l'accesso al file ed è una combinazione (un OR bit a bit) ha i seguenti predefiniti valori definiti nel file fcntl.h: O_APPEND, O_CREAT, O_EXCL, O_RDONLY, O_RDWR, O_WRONLY ecc., con i seguenti significati:

O_APPEND apre per aggiungere dati in fondo al file, e posiziona il puntatore alla posizione corrente alla fine del file
O_CREAT crea un file nuovo se non esiste
O_EXCL viene usato con O_CREAT, se il file già esiste causa errore
O_RDONLY apre e consente solo la lettura, si posiziona all'inizio del file
O_RDWR apre e consente sia lettura che scrittura si posiziona all'inizio
O_WRONLY apre e consente solo la scrittura
O_TRUNC se il file esiste viene troncato, riparte da lunghezza zero

Il parametro "**perms**" specifica i permessi da assegnare al file quando questo viene creato, in caso contrario viene ignorato.

Per creare un file si puo' anche usare la funzione:

int creat(char *filename, int perms)

Per chiudere un file si usa:

int close(int fd)

Per leggere/scrivere uno specificato numero di bytes da/su un file immagazzinati in una locazione di memoria specificata da "buffer" si utilizzano:

int read(int fd, char *buffer, unsigned length)
int write(int fd, char *buffer, unsigned length)

Queste due funzioni ritornano il numero di byte letti/scritti o -1 se falliscono.