GESTIONE DEI FILE

File come tipo di dati

Nel linguaggio C, i file vengono trattati come un tipo di dati derivato, cioè ottenuto dai tipi elementari esistenti.

In pratica, quando si apre e si gestisce un file, si ha a che fare con un puntatore al file, o file pointer, che è una variabile contenente un qualche riferimento univoco al file stesso.

Per gestire i puntatori ai file in C, occorre dichiarare una variabile come puntatore al tipo derivato FILE, come nell'esempio seguente:

```
#include <stdio.h>
...
main ()
{
  FILE *pf;
...
}
```

Il fatto che il nome utilizzato per questo tipo di dati, FILE, sia scritto utilizzando solo lettere maiuscole, lascia intendere che si tratta di una macro che si traduce in qualcosa di adatto al tipo particolare di piattaforma utilizzato.

Per questo stesso motivo, l'utilizzo di tale tipo richiede l'inclusione del file di intestazione stdio.h.

Apertura e chiusura

L'apertura dei file viene ottenuta normalmente con la funzione fopen() che restituisce il puntatore al file, oppure il puntatore nullo, NULL, in caso di fallimento dell'operazione.

L'esempio seguente mostra l'apertura del file mio_file contenuto nella directory corrente, con una modalità di accesso in sola lettura.

```
#include <stdio.h>
...
main ()
{
  FILE *pfMioFile;
  ...
  pfMioFile = fopen ("mio_file", "r");
  ...
}
```

Come si vede dall'esempio, si assegna il puntatore restituito dalla fopen(..) a una variabile adatta, che da quel momento identificherà il file, finché questo resterà aperto.

La chiusura del file avviene in modo analogo, attraverso la funzione fclose(..), che restituisce zero se l'operazione è stata conclusa con successo, oppure il valore rappresentato da EOF. L'esempio seguente ne mostra l'utilizzo.

```
fclose (pfMioFile);
```

La chiusura conclude l'attività sul file, dopo avere scritto tutti i dati eventualmente ancora rimasti in sospeso.

Normalmente, un file aperto viene definito come **flusso, o stream**;
così, nello stesso modo viene identificata la variabile puntatore che vi si riferisce.

In effetti, lo stesso file potrebbe anche essere aperto più volte con puntatori differenti, quindi è corretto distinguere tra file fisici su disco e file aperti, o flussi.

fopen()

La funzione fopen() permette di aprire il file indicato attraverso la stringa fornita come primo parametro, tenendo conto che tale stringa può contenere anche informazioni sul percorso necessario per raggiungerlo, secondo la modalità stabilita dal secondo parametro, anch'esso una stringa.

La sintassi è:

FILE *fopen (char *file, char *modalità)

La stringa fornita come secondo parametro, contenente l'informazione della modalità, può utilizzare i simboli seguenti:

- **r** apre il file in sola lettura, posizionandosi all'inizio del file;
- **r+** apre il file in lettura e scrittura, posizionandosi all'inizio del file;
- **w** apre il file in sola scrittura, creandolo se necessario, o troncandone a zero il suo contenuto se questo esisteva già;
- **w+** apre il file in scrittura e lettura, creandolo se necessario, o troncandone a zero il suo contenuto se questo esisteva già;
- **a** apre il file in scrittura in aggiunta (append), creandolo se necessario, o aggiungendovi dati a partire dalla fine e, di conseguenza, posizionandosi alla fine dello stesso;
- **a+** apre il file in scrittura in aggiunta e in lettura, creandolo se necessario, o aggiungendovi dati a partire dalla fine e, di conseguenza, posizionandosi alla fine dello stesso.

La funzione restituisce un puntatore al tipo FILE, riferito al file aperto, oppure si tratta del puntatore nullo (NULL) in caso di insuccesso.

fclose()

La funzione fclose() chiude il file rappresentato dal puntatore indicato come parametro della funzione.

Se il file era stato aperto in scrittura, prima di chiuderlo vengono scaricati i dati eventualmente accumulati nella memoria tampone (i buffer), utilizzando la funzione fflush().

int fclose (FILE *stream)

La funzione restituisce il valore zero se l'operazione si conclude normalmente, oppure EOF (fine del file) se qualcosa non funziona correttamente.

In ogni caso, il file non è più accessibile attraverso il puntatore utilizzato precedentemente.

EOF

Oltre alla macro FILE, è bene considerarne un'altra, anch'essa molto importante: EOF.

Si tratta di un valore, definito in base alle caratteristiche della piattaforma, utilizzato per rappresentare il raggiungimento della fine del file. Anche questa macro è definita all'interno del file stdio.h.

Lettura/Scrittura

L'accesso al contenuto dei file avviene generalmente solo a livello di byte e non di record.

Le operazioni di lettura e scrittura dipendono da un indicatore riferito a una posizione, espressa in byte, del contenuto del file stesso.

A seconda di come viene aperto il file, questo indicatore viene posizionato nel modo più logico, come è già stato descritto nella descrizione della funzione fopen().

Questo viene spostato automaticamente a seconda delle operazioni di lettura e scrittura che si compiono, tuttavia, quando di passa da una modalità di accesso all'altra, è necessario spostare l'indicatore attraverso le istruzioni opportune, in modo da non creare ambiguità.

La lettura avviene normalmente attraverso la funzione fread() che legge una quantità di byte trattandoli come un array. Per la precisione, si tratta di definire la dimensione di ogni elemento, espressa in byte, quindi la quantità di tali elementi. Il risultato della lettura viene inserito in un array, i cui elementi hanno la stessa dimensione. Si osservi l'esempio seguente:

```
...
char ac[100];
FILE *pf;
int i;
...
i = fread (ac, 1, 100, pf);
...
```

In questo modo si intende leggere 100 elementi della dimensione di un solo byte, collocandoli nell'array ac[], organizzato nello stesso modo.

Naturalmente, non è detto che la lettura abbia successo, o quantomeno non è detto che si riesca a leggere la quantità di elementi richiesta. Il valore restituito dalla funzione rappresenta la quantità di elementi letti effettivamente.

Se si verifica un qualsiasi tipo di errore che impedisce la lettura, la funzione si limita a restituire zero, o al limite, in certe versioni del linguaggio C, restituisce un valore negativo.

Quando il file viene aperto in lettura, l'indicatore interno viene posizionato all'inizio del file; quindi, ogni operazione di lettura sposta in avanti il puntatore, in modo che la lettura successiva avvenga a partire dalla posizione seguente:

```
...char ac[100];
FILE *pf;
int i;
...
pf = fopen ("mio_file", "r");
...
while (1) /* Ciclo senza fine */
{
  i = fread (ac, 1, 100, pf);
  if (i == 0)
  {
   break; /* Termina il ciclo */
  }
  ...
}
...
```

In questo modo, come mostra l'esempio, viene letto tutto il file a colpi di 100 byte alla volta, tranne l'ultima in cui si ottiene solo quello che resta da leggere.

La scrittura avviene normalmente attraverso la funzione fwrite() che scrive una quantità di byte trattandoli come un array, nello stesso modo già visto con la funzione fread().

```
...
char ac[100];
FILE *pf;
int i;
...
i = fwrite (ac, 1, 100, pf);
...
```

L'esempio, come nel caso di fread(), mostra la scrittura di 100 elementi di un solo byte, prelevati da un array. Il valore restituito dalla funzione è la quantità di elementi che sono stati scritti con successo. Se si verifica un qualsiasi tipo di errore che impedisce la scrittura, la funzione si limita a restituire zero, o al limite, in certe versioni del linguaggio C, restituisce un valore negativo.

Anche in scrittura è importante l'indicatore della posizione interna del file. Di solito, quando si crea un file o lo si estende, l'indicatore si trova sempre alla fine. L'esempio seguente mostra lo scheletro di un programma che crea un file, copiando il contenuto di un altro (non viene utilizzato alcun tipo di controllo degli errori).

```
#include <stdio.h>
#define BLOCCO 1024
...
main ()
{
  char ac[BLOCCO];
  FILE *pfIN;
  FILE *pfOUT;
  int i;
  ...
  pfIN = fopen ("fileIN", "r");
  ...
  pfOUT = fopen ("fileOUT", "w");
  ...
  while (1) /* Ciclo senza fine */
  {
    i = fread (ac, 1, BLOCCO, pfIN);
    if (i == 0)
   {
    break; /* Termina il ciclo */
   }
  ...
  fwrite (ac, 1, i, pfOUT);
  ...
  }
...
fclose (pfIN);
fclose (pfOUT);
...
}
```

fread() – fwrite()

fread():

size_t fread (void *buffer, size_t dimensione, size_t quantità, FILE *stream)

La funzione fread() permette di leggere una porzione del file identificato attraverso il puntatore riferito al flusso (stream), collocandola all'interno di un array. La lettura del file avviene a partire dalla posizione dell'indicatore interno al file, per una quantità stabilita di elementi di una data dimensione. La dimensione degli elementi viene espressa in byte e deve coincidere con la dimensione degli elementi dell'array ricevente, che in pratica si comporta da memoria tampone (buffer). L'array ricevente deve essere in grado di accogliere la quantità di elementi letti.

La funzione restituisce il numero di elementi letto effettivamente.

Il tipo di dati size_t serve a garantire la compatibilità con qualunque tipo intero, mentre il tipo void per l'array della memoria tampone, ne permette l'utilizzo di qualunque tipo.

fwrite():

size_t fwrite (void *buffer, size_t dimensione, size_t quantità, FILE *stream)

La funzione fwrite() permette di scrivere il contenuto di una memoria tampone (buffer), contenuta in un array, in un file identificato attraverso il puntatore riferito al flusso (stream). La scrittura del file avviene a partire dalla posizione dell'indicatore interno al file, per una quantità stabilita di elementi di una data dimensione. La dimensione degli elementi viene espressa in byte e deve coincidere con la dimensione degli elementi dell'array che rappresenta la memoria tampone. L'array in questione deve contenere almeno la quantità di elementi di cui viene richiesta la scrittura.

La funzione restituisce il numero di elementi scritti effettivamente.

Il tipo di dati size_t serve a garantire la compatibilità con qualunque tipo intero, mentre il tipo void per l'array della memoria tampone, ne permette l'utilizzo di qualunque tipo.

Indicatore interno

Lo spostamento diretto dell'indicatore interno della posizione i un file aperto è un'operazione necessaria quando il file è stato aperto sia in lettura che in scrittura e da un tipo di operazione si vuole passare all'altro. Per questo si utilizza la funzione fseek() ed eventualmente anche ftell() per conoscere la posizione attuale.

La posizione e gli spostamenti sono espressi in byte.

fseek() esegue lo spostamento a partire dall'inizio del file, oppure dalla posizione attuale, oppure dalla posizione finale. Per questo utilizza un parametro che può avere tre valori, rispettivamente 0, 1 e 2, identificati anche da tre macro: SEEK_SET, SEEK_CUR e SEEK_END. l'esempio seguente mostra lo spostamento del puntatore, riferito al flusso pf, in avanti di 10 byte, a partire dalla posizione attuale.

```
i = fseek (pf, 10, 1);
```

La funzione fseek() restituisce zero se lo spostamento avviene con successo, altrimenti si ottiene un valore negativo.

Lettura di un file a blocchi

L'esempio seguente mostra lo scheletro di un programma, senza controlli sugli errori, che, dopo aver aperto un file in lettura e scrittura, lo legge a blocchi di dimensioni uguali, modifica questi blocchi e li riscrive nel file.

```
#include <stdio.h>

/* ---------------------------------------------------------------- */

/* Dimensione del record logico */

/* ---------------------------------------------------------------- */

#define RECORD 10



main ()

{

char ac[RECORD];

FILE *pf;

int iQta;

int iPosizione1;

int iPosizione2;

pf = fopen ("mio_file", "r+"); /* lettura+scrittura */

while (1) /* Ciclo senza fine */

{

/* Salva la posizione del puntatore interno al file */

/* prima di eseguire la lettura. */

iPosizione1 = ftell (pf);

iQta = fread (ac, 1, RECORD, pf);

if (iQta == 0)

{

break; /* Termina il ciclo */

}

/* Salva la posizione del puntatore interno al file */

/* Dopo la lettura. */
```

```
iPosizione2 = ftell (pf);
/* Sposta il puntatore alla posizione precedente alla */
/* lettura. */
fseek (pf, iPosizione1, SEEK_SET);
/* Esegue qualche modifica nei dati, per esempio mette un */
/* punto esclamativo all'inizio. */
ac[0] = '!';
/* Riscrive il record modificato. */
fwrite (ac, 1, iQta, pf);
/* Riporta il puntatore interno al file alla posizione */
/* corretta per eseguire la prossima lettura. */
fseek (pf, iPosizione2, SEEK_SET);
}
fclose (pf);
}
```

fseek()

fseek():

int fseek (FILE *stream, long spostamento, int punto_di_partenza)

La funzione fseek() permette di spostare la posizione dell'indicatore interno al file a cui fa riferimento al flusso (stream) fornito come primo parametro. La posizione da raggiungere si riferisce al punto di partenza, rappresentato attraverso tre macro:

SEEK_SET rappresenta l'inizio del file;

SEEK_CUR rappresenta la posizione attuale;

SEEK_END rappresenta la fine del file.

Il valore dello spostamento, fornito come secondo parametro, rappresenta una quantità di byte che può essere anche negativa, indicando in tal caso un arretramento dal punto di partenza.

Il valore restituito da fseek() è zero se l'operazione viene completata con successo, altrimenti viene restituito un valore negativo: -1.

ftell()

ftell():

long ftell (FILE *stream)

La funzione ftell() permette di conoscere la posizione dell'indicatore interno al file a cui fa riferimento il flusso (stream) fornito come parametro. La posizione è assoluta, ovvero riferita all'inizio del file.

Il valore restituito in caso di successo è positivo, a indicare appunto la posizione dell'indicatore. Se si verifica un errore viene restituito un valore negativo: -1.

File di testo, fgets() – fputs()

I file di testo possono essere gestiti in modo più semplice attraverso due funzioni: fgets() e fputs(). Queste permettono rispettivamente di leggere e scrivere un file una riga alla volta, intendendo come riga una porzione di testo che termina con il codice di interruzione di riga.

La funzione fgets() permette di leggere una riga di testo di una data dimensione massima. Si osservi l'esempio seguente:

fgets (acz, 100, pf);

In questo caso, viene letta una riga di testo di una dimensione massima di 99 caratteri, dal file rappresentato dal puntatore pf. Questa riga viene posta all'interno dell'array acz[], con l'aggiunta di un carattere \0 finale. Questo fatto spiega il motivo per il quale il secondo parametro corrisponde a 100, mentre la dimensione massima della riga letta è di 99 caratteri. In pratica, l'array di destinazione è sempre una stringa, terminata correttamente.

Nello stesso modo funziona fputs(), che però richiede solo la stringa e il puntatore del file da scrivere. Dal momento che una stringa contiene già l'informazione della sua lunghezza perché possiede un carattere di conclusione, non è prevista l'indicazione della quantità di elementi da scrivere.

fputs (acz, pf);

fgets():

char *fgets (char *stringa, int dimensione, FILE *stream)

La funzione fgets() permette di leggere una stringa corrispondente a una riga di testo da un file. fgets() impone l'indicazione della dimensione massima della riga, dal momento che questa deve poi essere contenuta nell'array indicato come primo parametro. La dimensione massima, indicata come secondo parametro, rappresenta la dimensione dell'array di caratteri che deve riceverlo. Dal momento che per essere una stringa, questa deve essere terminata, allora la dimensione massima della riga sarà di un carattere in meno rispetto alla dimensione dell'array.

Pertanto, la lettura del file avviene fino al raggiungimento della dimensione dell'array (meno uno), oppure fino al raggiungimento del codice di interruzione di riga, che viene regolarmente incluso nella riga letta, oppure anche fino alla conclusione del file.

Se l'operazione di lettura riesce, fgets() restituisce un puntatore corrispondente alla stessa stringa (cioè l'array di caratteri di destinazione), altrimenti restituisce il puntatore nullo, NULL, per esempio quando il file ha raggiunto la fine.

fputs():

int fputs (const char *stringa, FILE *stream)

La funzione fputs() permette di scrivere una stringa in un file di testo. La stringa viene scritta senza il codice di terminazione finale, \0.

Il valore restituito è un valore positivo in caso si successo, altrimenti EOF.

270.6 I/O standard

Ci sono tre file che risultano aperti automaticamente:

standard input, corrispondente normalmente alla tastiera;

standard output, corrispondente normalmente allo schermo del terminale;

standard error, anch'esso corrispondente normalmente allo schermo del terminale.

Come è già stato visto in parte, si accede a questi flussi attraverso funzioni apposite, ma si potrebbe accedere anche attraverso le normali funzioni di accesso ai file, utilizzando per identificare i flussi i nomi: stdio, stdout e stderr.

Esempio

```
#include

main ()
{
 char ac[100];
 char ad[100];

 FILE *pf,*p1;

 int i;
 pf = fopen ("nuovo1.txt", "w");

 ac[0]='c';
 ac[1]='i';
 ac[2]='a';
 ac[3]='u';

 for (int j=0;j<100;j++)
    ad[j]=' ';

 fflush(stdin);
 i = fwrite (ac, 1, 4, pf);

// fseek(pf,-2,SEEK_END);

 fclose(pf);

 p1 = fopen ("nuovo1.txt", "r");
 fflush(stdin);
 i = fread (ad, 1, 4, p1);
 printf("%s", ad);

 fclose(p1);


 /* FILE *pfMioFile;
  pfMioFile = fopen ("nuovo1", "a+");
  fclose(pfMioFile);*/
  getchar();
  getchar();
}
```

Rinominare un file

```
//    Uso della funzione rename
//     int rename(const char *oldname, const char *newname);
//

#include

main ()
{
   char oldname[80], newname[80];

   /* chiede il nome del file da rinominare */

   printf("File da rinominare: ");
   gets(oldname);
   printf("Nuovo nome: ");
   gets(newname);

   /* Rinomina il file */

   if (rename(oldname, newname) == 0)
      printf("Renominato %s to %s.\n", oldname, newname);
   else
      perror("rename");

   getchar();
   getchar();

   return 0;
}
```