13.4 Lettura e scrittura formattata

Nel Capitolo 11 avevamo approfondito l'uso delle funzioni di libreria `printf` e `scanf`, che consentono di scrivere a video e di acquisire dati da tastiera, e introdotto le funzioni `fprintf` e `fscanf`; riprendiamo queste ultime in relazione alla scrittura e lettura su file:

- `fprintf` scrive sui file in modo formattato
- `fscanf` legge da file in modo formattato

La funzione `fprintf` si usa come nell'esempio seguente:

```
FILE *fp;
int i;
...
i = 2;
fprintf(fp, "Il valore di i è %d\n", i);
```

dove `fp` rappresenta il file sul quale si desidera scrivere; un uso analogo è quello che si fa di `fscanf`:

```
FILE *fp;
int i;
...
fscanf(fp,"%d", &i);
```

dove `fp` rappresenta il file dal quale si desidera leggere.
Altre due funzioni assai utili per poter operare sui file sono `fgets` e `fputs` che, rispettivamente, leggono e scrivono una riga su file, dove per riga si intende una sequenza di caratteri terminata dal carattere di newline ("`\n`"). La funzione `fgets` opera secondo questa falsariga:

```
char buf[100];
FILE *fp;
char *s;
int n;
...
n = 100;
s = fgets(buf, n, fp);
```

dove `buf` rappresenta il vettore in cui la riga letta dal file `fp` deve essere trasferita e `n` rappresenta la dimensione del vettore `buf`; la funzione `fgets` ritorna il puntatore a carattere: tale valore coincide con l'indirizzo di `buf` se tutto è andato a buon fine, mentre vale `NULL` nel caso in cui si sia verificato un errore o il file pointer si trovi posizionato a fine file.
La funzione `fgets` non si limita a trasferire una riga nel vettore `buf` ma aggiunge automaticamente a fine riga il carattere di fine stringa "`\0`".
La funzione `fputs` scrive una riga in un file. La modalità di uso è la seguente:

```
char buf[100];
FILE *fp;
...
fputs(buf,fp);
```

dove `buf` è il vettore che contiene la riga da scrivere sul file `fp` terminata con il carattere di fine stringa "`\0`".
Scriviamo ora un semplice programma di esempio che conta il numero di righe di un file (Listato 13.5).

```
/* Determinazione del numero di linee contenute in un file.
   Ogni linea è definita dal carattere di newline \n   */

#include <stdio.h>

main(int argc, char **argv)
{
char buf[100];
int linee;
FILE *fp;

if( argc < 2 )
  printf("Errato numero di parametri\n");
else {
  fp = fopen(argv[1], "r");             /* Apre il file */
  if(fp!= NULL) {                    /* Il file esiste? */
    linee = 0;        /* Inizializza contatore di linea */
    for(;;) {               /* Ciclo di lettura da file */
      if( fgets(buf,100,fp) == NULL )
        break;                           /* Fine file */
      linee++;            /* Incrementa contatore linee */
    }
    fclose(fp);                        /* Chiude il file */
    printf("Il file contiene %d linee\n", linee);
  }
  else
    printf("Il file %s non esiste\n",argv[1]);
}
}
```

Listato 13.5 Programma che conta il numero di righe di un file

Esistono inoltre due funzioni per leggere e scrivere un singolo carattere da file: `fgetc` e `fputc`.
La funzione `fgetc` opera nel seguente modo:

```
FILE *fp;
int c;
...
c = fgetc(fp);
```

dove `fp` rappresenta il file pointer. Il valore di ritorno della funzione `fgetc` è di tipo `int` e non di tipo `char` perché ha un duplice significato: se coincide con la costante simbolica `EOF`, definita nel file di include `stdio.h`, significa che il file pointer è posizionato a fine file e quindi nessun carattere è stato letto, mentre in caso contrario il valore di ritorno rappresenta il carattere letto dal file. La costante simbolica `EOF` è definita in modo tale che non possa mai eguagliare alcun carattere; solitamente vale –1.
La funzione `fputc` lavora nel seguente modo:

```
FILE *fp;
int c;
...
c = 'A;
fputc(c,fp);
```

dove `c` rappresenta il carattere da scrivere sul file pointer `fp`.
Scriviamo ora un programma che conta il numero di caratteri numerici contenuti in un file (Listato 14.6).

```
/* Determinazione del numero di caratteri numerici
   (cifre decimali) presenti in un file         */

#include <stdio.h>

main(int argc, char **argv)
{
FILE *fp;
int c;
int nc;

if( argc < 2 )
  printf("Errato numero di parametri\n");
else {
  fp = fopen(argv[1],"r");                  /* Apre il file */
  if(fp!=NULL) {                          /* Il file esiste? */
    nc = 0;                     /* Inizializza il contatore */
    while((c = fgetc(fp)) != EOF)       /* Ciclo di lettura */
      if(c>='0' && c<='9') nc++; /* Incrementa il contatore */
    fclose(fp);                           /* Chiude il file */
    printf("Il numero di caratteri numerici è: %d\n", nc);
  }
  else
    printf("Il file %s non esiste\n",argv[1]);
}
}
```

Listato 13.6 Programma che conta il numero di caratteri numerici contenuti in un file

Spesso il programmatore desidera sapere semplicemente se il file pointer si trova posizionato a fine file; a tale scopo ha a disposizione la funzione `feof`, la cui modalità d'uso è:

```
int ret;
FILE *fp;
...
ret = feof(fp);
```

dove `fp` rappresenta il file pointer. Il valore di ritorno `ret` conterrà il numero 1 se il file pointer è posizionato a fine file e il numero 0 in caso contrario.
Tutte le funzioni sin qui descritte lavorano in modalità bufferizzata: molte operazioni di scrittura non sono fisicamente eseguite su disco ma rimangono temporaneamente memorizzate in memoria centrale per ottimizzare le prestazioni. Soltanto una scrittura su molte viene realmente eseguita. Lo stesso discorso vale per le operazioni di lettura, poiché il C durante le letture fisiche da disco cerca di trasferire in memoria centrale una quantità di informazioni superiore a quella richiesta dal programmatore, con l'obiettivo di evitare altre letture fisiche successive, poiché i dati che verranno richiesti sono già stati pre-letti.
Il C esegue queste operazioni in maniera del tutto trasparente, utilizzando dei buffer di memoria la cui dimensione è ottimizzata sulla base delle caratteristiche del sistema operativo ospite e della memoria di massa. Al momento della chiusura di un file o quando il programma termina la sua esecuzione, il contenuto di tutti i buffer viene scaricato su disco. La costante simbolica `BUFSIZ`, contenuta nel file di include `stdio.h`, rappresenta la dimensione in byte dei buffer per le operazioni di lettura e scrittura.
Talvolta il programmatore ha la necessità di scaricare su disco tutte le scritture che sono rimaste temporaneamente memorizzate nei buffer. La funzione `fflush` assolve il compito di scaricare il contenuto del buffer associato a un file pointer:

```
#include <stdio.h>

main()
{
FILE *fp;

fp = fopen("ordini", "w");
fprintf(fp,"Giovanni Fuciletti");

fflush(fp);
...
```

Nel caso in cui il programmatore desideri disabilitare la bufferizzazione su di un file pointer può usare la funzione `setbuf`:

```
FILE *fp;
...
setbuf(fp, NULL);
```

dove `fp` è il file pointer, mentre il secondo parametro `NULL` richiede la disabilitazione completa della bufferizzazione.