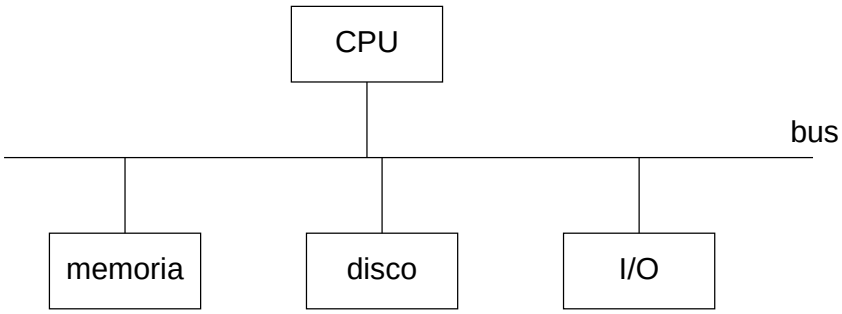
File di testo

I file (archivi) sono sequenze di dati su dispositivi di memorizzazione di massa, ossia dispositivi che consentono di immagazzinare grandi quantità di dati in modo permanente. I tipici dispositivi hardware di memorizzazione di file sono i dischi (fissi e mobili). In queste pagine si vedrà in che modo si possono utilizzare i file di testo, ossia i file in cui i dati sono memorizzati come sequenze di caratteri.

Memoria primaria e secondaria

Memoria primaria e secondaria

La struttura di un calcolatore, ad alto livello, si può rappresentare come il diagramma qui sotto:



Il bus è un meccanismo di comunicazione fra i vari elementi componenti. Quello di sopra è un modello molto semplificato della struttura hardware di un calcolatore. Le varie parti che compongono il calcolatore sono:

cpu
: la parte che effettua calcoli ed elaborazioni

memoria
: dispositivo di memorizzazione veloce, non permanente, di piccola dimensione (memoria primaria)

disco
: memoria, più lento, permanente, di grande dimensione (memoria secondaria)

I/O
: dispositivi di ingresso e uscita: tastiera, mouse, schermo, ecc.

Le variabili di un programma sono memorizzate nella memoria (primaria) del calcolatore. Questo significa che si può accedere ad esse in lettura e scrittura molto velocemente. Gli svantaggi sono: la memoria primaria è (relativamente) piccola, per cui non si possono memorizzare grandi quantità di dati; la memoria primaria è inoltre non permanente, per cui i valori delle variabili non sono permanenti.

Per questo motivo, se vogliamo memorizzare dati in modo che non vengano cancellati allo spegnimento del calcolatore, oppure dobbiamo usare una grande quantità di dati, occorre memorizzarli nella memoria secondaria, ossia sui dischi.

Apertura e chiusura di un file

Apertura e chiusura di un file

I dati memorizzati su disco sono divisi in file. Ogni file è un insieme di dati, ed è caratterizzato da un nome. Un nome di file è semplicemente una stringa.

Un programma C può leggere i dati memorizzati in un file. Serve naturalmente sapere il nome del file che si vuole leggere. La prima cosa da fare, quando si vuole leggere un file, è quello di dire il nome del file da leggere. Questo viene fatto con la funzione `fopen`. Questa funzione ha due parametri: il primo è il nome del file che si vuole leggere (i nomi di file sono stringhe, ossia vettori di caratteri). Il secondo argomento è una stringa che indica il modo in cui si vuole acceddere al file (vedi più avanti); per leggere il file, questa stringa deve valere `"r"`, ossia deve essere la stringa composta dal solo carattere `r`, più il terminatore di stringa. Il prototipo della funzione `fopen` è questo:

```
FILE *fopen(char *nome, char *modo);
```

La chiamata alla funzione `fopen` viene detta *apertura* del file. Il valore che viene ritornato da questa funzione è di tipo `FILE *`, e viene detto descrittore di file. Il descrittore di file è un valore che viene usato dalle successive funzioni di accesso al file, per indicare di quale file si sta parlando.

Per dire che abbiamo finito di leggere il file, e non ci occorre più, usiamo la funzione `fclose`, che *chiude* il file. Questa funzione prende come argomento il descrittore del file che si vuole chiudere, ossia valore che è stato ritornato dalla `fopen`.

Il programma aprichiudi.c apre un file, e lo chiude immediatamente dopo. Non è evidentemente di molta utilità: serve solo a far vedere come si apre e chiude un file.

```
/*
  Apre un file e poi lo chiude.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;

                /* apre il file */
  fd=fopen("test.txt", "r");

                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Questo è il primo uso di un descrittore di file: dal momento che può aprire più file (questo avviene se serve leggere dati da più di un file), quando uno di questi non serve più, occorre chiuderlo dicendo però quale dei file va chiuso. Questo concetto verrà chiarito più avanti.

Nota: le variabili di tipo `FILE *` sono puntatori. Tuttavia, non è necessario creare una zona di memoria con `malloc`: questo viene fatto dalla funzione `fopen`.

Verifica in apertura

Verifica in apertura

Cosa succede se si apre un file che non esiste su disco? Questo può succedere per esempio se si sbaglia a digitare il nome del file: non è detto che sul disco ci sia un file con il nome che è stato passato a `fopen`.

Esistono poi altri errori che possono verificarsi quando si cerca di accedere a un file. Per il momento, non ci interessano. Quello che conta è il modo in cui il programma può verificare se c'è stato un errore in apertura oppure no.

La regola è semplicemente che, se si è verificato un errore in apertura del file, il valore che viene ritornato da `fopen` è la costante `NULL`. Quindi, se dopo aver aperto il file vogliamo verificare se ci sono stati errori, dobbiamo confrontare il valore ritornato da `fopen` con `NULL`.

Il seguente programma verifica.c apre un file, e poi verifica se ci sono stati errori. In caso di errore, il valore del descrittore di file è `NULL`. In questo caso, si stampa un messaggio di errore e si termina l'esecuzione del programma.

```
/*
  Apre il file, e verifica.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;

                /* apre il file */
  fd=fopen("test.txt", "r");

                /* verifica errori in apertura */
  if( fd==NULL ) {
    printf("Si e' verificato un errore in apertura del file\n");
    exit(1);
  }

                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Questo programma stampa un messaggio se si è verificato un errore durante l'apertura del file, ossia se non siamo riusciti ad aprire il file. Però non dice quale specifico errore si è verificato. Per fare questo, usiamo la funzione `perror`, che stampa un messaggio che indica quale specifico errore si è verificato. Questa funzione prende una stringa, che viene stampata prima del messaggio di errore. Il programma perror.c è simile al precedente, ma il messaggio di errore viene stampato con `perror`.

```
/*
  Apre il file, e verifica.
*/

#include#include
int main() {
  FILE *fd;
```

```
                /* apre il file */
  fd=fopen("test.txt", "r");

                /* verifica errori in apertura */
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }

                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Lettura di un file

Lettura di un file

La funzione `fopen` serve a dire che vogliamo leggere dati da un file, ma non fa nessuna lettura. Per leggere da file si usa la funzione `fscanf`. Questa funzione è molto simile alla funzione di input da tastiera `scanf`, ma legge da file. Il suo primo argomento è un descrittore di file, e indica da quale file vogliamo leggere.

Per esempio, per leggere un intero da tastiera usiamo una istruzione del tipo:

```
  scanf("%d", &x);
```

dove `x` è una variabile intera. Per leggere da file, usiamo una istruzione simile, in cui al posto di `scanf` mettiamo `fscanf`, che ha un argomento iniziale aggiuntivo, che è il descrittore del file da cui vogliamo leggere:

```
  fscanf(fd, "%d", &x);
```

Il descrittore di file `fd` deve essere di tipo `FILE *`, e deve essere il valore che è stato ritornato dalla funzione `fopen`. Quindi, per leggere un intero da file, occorre:

1. aprire il file: questo ritorna un descrittore di file che va memorizzato in una variabile
2. usare `fscanf` per leggere un intero dal file identificato dal descrittore di file
3. chiudere il file

Il seguente programma intero.c segue questo metodo per leggere un numero intero da un file che si chiama `"intero.txt"`.

```
/*
  Legge un intero da file.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;
  int x;


                /* apre il file, legge il numero, chiude */
```

```
  fd=fopen("intero.txt", "r");

  fscanf(fd, "%d", &x);

  fclose(fd);


               /* stampa il numero */

  printf("Il numero letto da file e' %d\n", x);

  return 0;
}
```

Questo programma non fa nessuna verifica che il file sia stato aperto correttamente. Per effettuare queta verifica, basta controllare che il descrittore di file ritornato da `fopen` sia diverso da `NULL`. Il programma interover.c fa questa verifica.

Una considerazione finale: una volta letto un valore da file, questo valore rimane ovviamente memorizzato nella variabile anche dopo che il file è stato chiuso. Nell'esempio di sopra, una volta letto il numero `x` si può anche chiudere il file, e il valore nella variabile `x` rimane inalterato (fino a che non viene esplicitamente riassegnato).

Scrittura su file

Scrittura su file

Per scrivere su un file, occorre specificare che il file va aperto per scrivere. In altre parole, dobbiamo dire che vogliamo scrivere sul file già nel momento in cui il file viene aperto: non è possibile aprire il file in lettura e poi ‘‘cambiare idea’’ e scrivere sul file.

Per aprire un file in scrittura, si usa sempre la funzione `fopen`, passando come primo argomento il nome del file da aprire, e come secondo argomento la stringa `"w"`. Per esempio, per dire che vogliamo scrivere sul file `scrivi.txt`, usiamo la istruzione:

```
  fd=fopen("scrivi.txt", "w");
```

Come nel caso della lettura, occorre memorizzare il valore di ritorno della funzione (il descrittore di file), perchè è quello che serve per identificare il file nelle successivo operazioni (scrittura e chiusura).

La funzione che si usa per scrivere su un file è `fprintf`. Il suo primo argomento è il descrittore del file su cui si vuole scrivere. I successivi argomenti sono gli stessi della `printf`. L'unica differenza è che la funzione `fprintf` scrive sul file invece che sullo schermo. Quindi, dato che per scrivere un numero intero su schermo si usa la istruzione:

```
  printf("%d\n", x);
```

allora per scrivere un intero su un file il cui descrittore è `fd` facciamo:

```
  fprintf(fd, "%d\n", x);
```

Il seguente programma scrivi.c apre il file di nome `scrivi.txt`, ci scrive un intero, e lo chiude.

```
/*
  Scrive un intero su file
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;
  int x=-32;

                /* apre il file in scrittura */
  fd=fopen("scrivi.txt", "w");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* scrive il numero */
  fprintf(fd, "%d\n", x);


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

La funzione `fopen` si comporta in modo diverso a seconda se il file che si sta aprendo in scrittura esiste oppure no.

il file non esiste
: è chiaro che non si deve generare nessun errore (altrimenti non si potrebbero mai creare nuovi file); il file viene creato di dimensione zero (la dimensione aumenta quando si fanno le successive operazioni di scrittura);

il file esiste già
: in questo caso il contenuto del file viene azzerato; in altre parole, è come se il file venisse cancellato e ricreato vuoto.

In entrambi i casi, se si apre un file in scrittura e poi lo si chiude subito dopo, quello che si ottiene è un file di dimensione zero.

Scrittura in coda a un file

Scrittura in coda a un file

Fino ad ora, si sono visti solo due modi di accedere a un file: in lettura e in scrittura. Esiste un terzo modo, che in effetti è un secondo metodo di apertura in scrittura.

Come si è visto, facendo `fopen(..., "w")`, se il file esiste il suo contenuto viene azzerato, e si comincia a scrivere dall'inizio. È possibile fare in modo che la scrittura avvenga di seguito al file, piuttosto che all'inizio. La modalità *append* permette appunto di aprire un file in maniera che sia possibile scrivere in coda al contentuto attuale del file:

```
  fd = fopen("....", "a");
```

Si tratta quindi semplicemente di passare la stringa `"a"` come secondo argomento alla funzione `fopen`. L'effetto di questa istruzione è che risulta ora possibile scrivere sul file usando la funzione `fprintf`.

Per chiarire meglio la differenza fra la modalitè `w` e la modalità `a`, vediamo cosa succede se un file, che già esiste, viene aperto in scrittura nei due casi, e poi subito dopo chiuso.

`w` il file viene azzerato, ossia il suo contenuto viene cancellato; alla fine della chiamata alla funzione `fopen`, la sua dimensione è 0;

`a` il file viene aperto in scrittura in fondo, ossia tutte le cose che verranno scritte andranno ad aggiungersi a quelle che già stanno sul file; quindi, la operazione di apertura non modifica il contenuto del file; se il file viene chiuso subito dopo, non subisce modifiche.

La differenza fra le due modalità di apertura è quindi che, nel primo caso, l'intero contenuto del file viene cancellato, e le operazioni di scrittura vanno a scrivere sul file dall'inizio; al contrario, il secondo modo (append) lascia inalterato il contenuto del file, e le operazioni di scrittura aggiungono dati in fondo al file.

Il programma append.c apre un file in append, e scrive un numero in coda al file. L'effetto è quello di aggiungere il numero in fondo al file. Si noti che la istruzione che si usa per scrivere sul file è sempre la istruzione `fprintf`. La differenza fra questo programma e quello che scrive su file cancellando il contenuto precedente scrivi.c non è nella istruzione di scrittura, ma solo nel modo in cui il file viene aperto. In altre parole, per scrivere si usa comunque l'istruzione `fprintf`, esattamente nello stessp modo; la differenza fra scrittura dall'inizio del file e scrittura in coda sta solo nel modo in cui il file viene aperto.

```
/*
  Scrive un intero in coda a un file
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;
  int x=-32;

                /* apre il file in scrittura */
  fd=fopen("scrivi.txt", "a");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* scrive il numero */
  fprintf(fd, "%d\n", x);


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Anche nel caso in cui il file viene aperto in append mode, la istruzione `fopen` può ritornare un errore. Per rilevare possibili errori, va messa una istruzione condizionale per controllare se l'operazione di apertura ha avuto successo.

Come nel caso della apertura in scrittura, il file viene creato (vuoto) se non esiste. Quindi, se non esiste un file il cui nome è quello passato come primo argomento alla funzione `fopen`, allora non c'è nessuna differenza fra l'apertura in scrittura e l'apertura con scrittura in coda.

Lettura e scrittura in sequenza

Lettura e scrittura in sequenza

Le operazioni di lettura e scrittura `fscanf` e `fprintf` hanno un comportamento simile alle funzioni `scanf` e `printf` con la differenza che la lettura e scrittura avviene da file invece che da tastiera/schermo.

Questa regola permette di capire il comportamento delle funzione `fscanf` e `fprintf` in situazioni più complesse. Cosa succede per esempio se si fanno due operazioni `fscanf(fd, "%d", &x)` l'una dopo l'altra? Basta pensare cosa succederebbe se al posto della `fscanf` ci fosse una `scanf`. L'effetto sarebbe quello di leggere due interi, l'uno dopo l'altro, da tastiera. Dal momento che stiamo invece usando la `fscanf`, facciamo la stessa cosa ma leggendo da file. Quindi, si leggono due interi, l'uno dopo l'altro, dal file il cui descrittore è `fd`.

È come se le operazioi di lettura avvenissero da tastiera invece che da file, e sulla tastiera venisse digitato il contenuto del file. Quindi, dopo che si è letto qualcosa, la successiva operazione di lettura legge quello che segue sul file. In altre parole, due operazioni `fscanf` in sequenza non leggono la stessa cosa: la seconda legge quello che resta sul file dopo aver letto la prima.

Il programma leggidue.c è un programma in cui c'è una sequenza di due operazioni di lettura da file.

```
/*
  Legge due interi da file
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  int x, y;
  FILE *fd;

                /* apre il file */
  fd = fopen("dueinteri.txt", "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* legge il primo intero */
  fscanf(fd, "%d", &x);


                /* legge il secondo intero */
  fscanf(fd, "%d", &y);
```

```
                /* chiude il file */
  fclose(fd);


                /* stampa i due interi */
  printf("Ho letto %d e %d\n", x, y);

  return 0;
}
```

La prima cosa che avviene eseguendo questo programma è che il file viene aperto in lettura. Dopo il controllo se l'apertura ha avuto successo, viene letto un intero con la istruzione `fscanf(fd, "%d", &x)`. La successiva operazione è ancora una istruzione di lettura da file `fscanf(fd, "%d", &y)`. Da notare che questa seconda operazione non legge lo stesso intero della prima. Al contrario, viene letto il secondo intero che si trova su file (se c'è). Per provare questo programma, occorre creare un file di testo di nome `dueinteri.txt`, in cui vanno scritti due numeri interi.

Per quello che riguarda la scrittura su file, questa avviene sempre in modo consecutivo. Dopo che si è scritto qualcosa usando `fprintf`, le successive operazioni di scrittura scrivono di seguito. In altre parole, se si usa due volte di seguito la `fprintf`, la due cose scritte su file appaiono l'una di seguito all'altra. Il programma seguente scrividue.c scrive un intero su file con l'istruzione `fprintf(fd, "%d ", x)`. La successiva istruzione di stampa su file `fprintf(fd, "%d ", y)` scrive un secondo intero di seguito al primo. L'effetto complessivo è che sul file vengono scritti i due numeri di seguito.

```
/*
  Scrive due interi da file
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  int x, y;
  FILE *fd;

  x=-10;
  y=21;

                /* apre il file */
  fd = fopen("dueinteri.txt", "w");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* scrive il primo intero */
  fprintf(fd, "%d ", x);


                /* scrive il secondo intero */
  fprintf(fd, "%d ", y);


                /* chiude il file */
```

```
  fclose(fd);

  return 0;
}
```

Si noti che, per essere sicuri di avere uno spazio di separazione fra i due interi, occorre metterlo esplicitamente. È per questo che la stringa di formato della prima istruzione di stampa è `"%d "` con uno spazio in fondo, invece che `"%d"`. In questo secondo caso, i due interi sarebbero stati scritti su file consecutivamente, ossia senza spazi in mezzo. Questo avrebbe prodotto un file con contenuto `-1021`, senza spazi fra i due numeri. Un file con questo contenuto viene di solito interpretato come un file che contiene un solo numero intero di valore -1021. Questo è anche quello che si ottiene leggendo un intero da file usando la funzione `fscanf`. Si può quindi dire che non mettere spazi fra due interi quando di scrivono su file è un errore.

Lettura di un array da file

Lettura di un array da file

Usando le funzioni di apertura, lettura e scrittura da file, e avendo visto come si scrive/legge in sequenza, possiamo ora vedere operazioni più complesse. Supponiamo quindi di avere un file di nome `array.txt` che viene usato per memorizzare un array.

Più precisamente, il file contiene una sequenza di numeri interi. Il primo numero rappresenta la dimensione dell'array, ossia il numero di elementi che contiene. Questi elementi sono memorizzati su file di seguito, in sequenza.

Vogliamo un programma che legga un file di questo genere, e memorizzi i dati che contiene in un vettore. Facciamo l'ipotesi che ci siano al massimo 100 elementi. Questo permette di usare un vettore dichiarato staticamente per contentere i dati scritti su file.

Dopo l'operazione di apertura del file, con controllo errori, quello che serve è leggere il primo numero intero scritto su file, che è il numero di elementi successivi del file. Per memorizzare questo numero, dichiariamo una variabile intera `n`. Facciamo quindi una operazione di lettura di intero da file:

```
  fscanf(fd, "%d", &n);
```

A questo punto sappiamo che sul file ci sono altri `n` interi, che vanno letti da file e messi in ordine in un vettore. Dichiariamo una variabile `vett` come vettore di 100 interi. Per leggere gli elementi del vettore da file, facciamo un ciclo, in cui leggiamo un elemento a ogni iterazione.

```
    for(i=0; i<=n-1; i++)
      fscanf(fd, "%d", &vett[i]);
```

Il problema è risolto. Il vettore contiene ora gli elementi memorizzati sul file, e la variabile `n` indica quanti interi sono stati effettivamente letti, ossia quanti elementi del vettore contengono effettivamente dei valori che sono stati letti da file (gli eltri elementi del vettori non sono stati inizializzati o letti, quindi non contengono valori significativi).

Il programma completo è leggiarray.c, che contiene anche un controllo aggiuntivo se `n` è effettivamente minore di 100. Dopo aver letto il vettore, stampa i suoi elementi su schermo.

```
/*
  Legge un array da file.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  int n;
  int vett[100];
  FILE *fd;
  int i;

                /* apre il file */
  fd=fopen("array.txt", "r");

                /* verifica errori in apertura */
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }

                /* legge il numero di elementi del vettore */
  fscanf(fd, "%d", &n);


                /* legge l'array */
  if(n>=100)
    printf("Troppi elementi da leggere\n");
  else
    for(i=0; i<=n-1; i++)
      fscanf(fd, "%d", &vett[i]);


                /* stampa l'array */
  for(i=0; i<=n-1; i++)
    printf("%d\n", vett[i]);

                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Scrivere la media in fondo a un file

Scrivere la media in fondo a un file

Questo esercizio consiste nel leggere di un certo numero di interi da file, calcolare la loro media, e scrivere la media in coda al file.

L'esercizio si risolve facilmente combinando il programma di lettura di un array da file con il programma di scrittura di un intero in coda a un file. Quello che occorre, infatti, è leggere un certo numero di elementi da file (supponiamo che il primo intero rappresenti il numero di interi successivi su file), come nel caso del programma di lettura di un array da file, e poi di scrivere un intero in coda a un file, come nel caso del programma di scrittura in coda a un file.

Il programma che risolve questo problema media.c, legge un vettore da file, e chiude il file. A questo punto il vettore contiene gli elementi del file. Possiamo quindi calcolare la loro media. Per scrivere questo valore in fondo al file, lo apriamo di nuovo in append mode, e scriviamo il valore della media. Dato che il file è stato aperto in append mode, le operazioni di scrittura aggiungono in fondo al file, lasciando inalterato il contenuto precedente.

```
/*
  Scrive la media degli interi di un file
in fondo al file stesso.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  int n;
  int vett[100];
  FILE *fd;
  int i;
  int s, m;


                /* apre il file */
  fd=fopen("array.txt", "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }

                /* legge il numero di elementi del vettore */
  fscanf(fd, "%d", &n);


                /* legge l'array */
  if(n>=100)
    printf("Troppi elementi da leggere\n");
  else
    for(i=0; i<=n-1; i++)
      fscanf(fd, "%d", &vett[i]);


                /* chiude il file */
  fclose(fd);


                /* calcola la media */
  s=0;
  for(i=0; i<=n-1; i++)
    s+=vett[i];
  m=s/n;


                /* riapre il file in append */
  fd=fopen("array.txt", "a");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* scrive la media (in fondo al file) */
```

```
  fprintf(fd, "%d\n", m);


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

In questo programma si vede che è possibile aprire e chiudere lo stesso file più volte all'interno dello stesso programma. Si può anche usare la stessa variabile per memorizzare il descrittore di file. Infatti, se il file viene chiuso, il valore contenuto nella variabile non serve più a niente, dato che il file è stato chiuso.

Rilevazione end-of-file

Rilevazione end-of-file

Il programma di lettura di array da file non contiene nessun meccanismo di rilevazione di errori su file. Per esempio, non si accorge se il file è più corto del dovuto, ossia contiene meno elementi di quelli specificati come primo intero. Inoltre, non contiene nessun controllo sul fatto che il file potrebbe contenere dei caratteri, che quindi non sono interpretabili come interi. A seconda di come è specificato il formato del file da cui leggere, queste situazioni possono venire considerate o meno degli errori da parte di chi ha scritto il file. Per il momento, assumiamo che entrambe le situazioni vanno interpretate come errori.

Il modo in cui le funzioni di accesso a file comunicano un eventuale errore è attraverso il valore di ritorno. Per esempio, errori in apertura del file si rilevano mettendo il valore di ritorno della funzione `fopen` in una variabile, e controllando il valore di questa variabile. Per le operazioni di lettura da file si usa lo stesso metodo: `fscanf` è in effetti una funzione che ritorna un valore intero, che indica se ci sono stati o meno degli errori in scrittura. Per controllare se ci sono stati errori, dobbiamo quindi memorizzare il valore di ritorno della funzione `fscanf` in una variabile, e poi controllare se il valore di questa variabile indica se ci sono stati errori.

Per il momento, diciamo che la funzione `fscanf` ritorna il valore `EOF` nel caso in cui si è tentato di leggere qualcosa da un file, ma il file è già finito. In altre parole, se si è letto tutto il contenuto di un file, e si usa `fscanf` per leggere ancora, questa funzione ritorna il valore `EOF`. Questo `EOF` è una macro di cui non ci interessa il valore.

Si noti che `EOF` è il *valore di ritorno* della funzione `fscanf`, e non il valore che viene memorizzato nelle variabili da leggere. In altre parole, se si vuole leggere un intero da file, controllando che non si sia arrivati alla fine del file, occorre controllare se il valore di ritorno di `fscanf` vale `EOF`, con un frammento di codice come quello che segue:

```
    res=fscanf(fd, "%d", &x);
    if( res==EOF ) {
      printf("Non riesco a leggere un intero: il file e' finito\n");
      exit(1);
    }
```

Le variabili `res` e `x` sono entrambe dichiarate come intere. Mentre `x` contiene il valore che si è letto da file (in caso la cosa sia avvenuta con successo) la variabile `res` viene usata per memorizzare il valore di ritorno di `fscanf`, per cui viene usata per controllare che non ci siano stati errori in lettura.

Il seguente programma arrayeof.c legge un vettore da file. Ci si ferma solo nel caso in cui si sia arrivati a leggere cento elementi, oppure quando si cerca di leggere un intero ma il file è terminato. Si noti che, in questo caso, il numero iniziale che indica quanti interi ci sono su file non è necessario. Infatti, il numero di elementi letti viene determinato leggendo via via gli interi da file, incrementando ogni volta il numero di elementi letti fino a che non si arriva alla fine del file.

In questo caso, se si incontra la fine del file, non si deve stampare un messaggio di errore. Infatti, la fine del file indica semplicemente che non c'è altro da leggere, ossia che siamo arrivati all'ultimo elemento del vettore. In questo caso, la fine del file viene usata per indicare quanti elementi da leggere ci sono. Al contrario, nel caso in cui si fosse per esempio specificato che il file deve contenere esattamente cento elementi, allora trovare `EOF` prima del centesimo elemento sarebbe stato un errore. Si può dire che incontrare la fine di un file durante una operazione di lettura può essere o non essere un errore, a seconda di come è stato specificato il formato del file.

```
/*
  Legge un array da file, con rilevazione di end-of-file.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  int n;
  int vett[100];
  FILE *fd;
  int res;
  int i;

                /* apre il file */
  fd=fopen("arrayeof.txt", "r");

                /* verifica errori in apertura */
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* legge l'array */
  for(n=0; n<=100-1; n++) {
    res=fscanf(fd, "%d", &vett[n]);
    if( res==EOF )
      break;
  }


                /* stampa l'array */
  for(i=0; i<=n-1; i++)
    printf("%d\n", vett[i]);

                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Rilevazione errori

Rilevazione errori

Oltre alla fine del file, esistono altri errori possibili quando si cerca di leggere da file. Alcuni errori sono dovuti per esempio a problemi di livello hardware, altri a livello di sistema operativo, ecc. Di questo tipo di errori non ci interessiamo.

Un errore di cui invece parliamo è il fallimento di conversione. Quando si cerca di leggere da file un intero, la funzione `fscanf` deve convertire una stringa in intero. Infatti, un file di testo è una sequenza di caratteri, ossia una stringa. Se vogliamo un intero, dobbiamo convertire i caratteri che si trovano su file in un intero.

Questa conversione non è sempre possibile. Si pensi per esempio alla situazione in cui su file sono memorizzati i caratteri `abe mds`. Questa sequenza non corrisponde a nessun intero. Se si cerca di leggere un intero, deve essere possibile capire che la conversione da stringa a intero è fallita.

Errori di questo genere si riflettono sul valore di ritorno della funzione `fscanf`. Questa funzione, in generale, può leggere più di un oggetto per volta. Per esempio, può leggere, con una chiamata sola, un intero, una stringa, e un reale. Il valore di ritorno della funzione è il numero di oggetti che sono è stato possibile convertire. Il valore `EOF` viene ritornato solo nel caso in cui si è incontrata la fine del file prima ancora di riuscire a convertire anche un solo intero.

> la funzione `fscanf` ritorna il numero di oggetti che è stato possibile leggere (convertire da stringa al tipo dell'oggetto), oppure `EOF` se il file è finito prima di poter leggere il primo oggetto.

Esempio:

`fscanf(fd, "%d %d", &x, &y);` ha i seguenti possibili valori di ritorno:

2
: sono stati letti (convertiti con successo) due interi, memorizzati nell varabili `x` e `y`;

1
: un solo intero è stato letto con successo, ed è stato memorizzato nella variabile `x`; non è stato possibile leggere il secondo intero: questo può essere dovuto al fallimento di conversione (quello che sta scritto su file non si può convertire in intero), oppure al fatto che è stata incontrata la fine del file;

0
: non è stato possibile leggere nessuno dei due numeri interi: questo è dovuto alla presenza di caratteri non interpretabili come interi su file;

`EOF`
: nessun intero è stato letto da file, perchè si è incontrata la fine del file prima ancora di riuscire a leggere il primo intero.

Si noti che, se il risultato è 1, allora non è possibile capire se il problema è stato un errore di conversione oppure la fine del file. Per il momento, questo fatto non ci interessa, visto che leggiamo sempre un unico elemento per volta. Nel caso in cui la `fscanf` viene usata per leggere da file il valore di una sola variabile, allora i possibili valori di ritorno sono:

`fscanf(fd, "%d", &x);`

1
: un intero è stato letto e memorizzato in `x`;

0
: non è stato possibile leggere l'intero, perchè quello che sta scritto su file non è interpretabile come un intero;

`EOF`
: si è incontrata la fine del file prima di poter leggere un intero da memorizzare in `x`.

Da questa schema è chiaro che, nel caso di lettura di un solo intero, è sempre possibile capire quale errore si è verificato semplicemente guardando il valore di ritorno della funzione `fscanf`.

Il programma arrayerr.c è l'ultima versione del programma di lettura di un array da file. Questa volta è stato aggiunto un controllo sulla lettura: se la conversione in intero è fallita allora la funzione `fscanf` ritorna il valore 0, e in questo caso il programma termina.

```
/*
  Legge un array da file fino all'eof.
Salta le cose che non riesce a leggere.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  int n;
  int vett[100];
  FILE *fd;
  int res;
  int i;

                /* apre il file */
  fd=fopen("arrayerr.txt", "r");

                /* verifica errori in apertura */
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* legge l'array */
  for(n=0; n<=99; n++) {
    res=fscanf(fd, "%d", &vett[n]);
    if( res==EOF )
      break;
    else if( res==0 ) {
      printf("Non riesco a leggere un intero da file\n");
      exit(1);
    }
  }


                /* stampa l'array */
  for(i=0; i<=n-1; i++)
    printf("%d\n", vett[i]);

                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Nel caso in cui non interessa stabilire quale particolare errore si è verificato, basta controllare che il valore di ritorno della funzione `fscanf` coincida con il numero di elementi che doveva leggere (in questo caso, uno), e generare un messaggio di errore in caso contrario.

Sottrazione fra file

Lo scopo di questo esercizio è quello di leggere due file che contengono degli interi, e scrivere un file che contiene le differenza fra elementi del primo e del secondo.

Per essere più precisi, i due file contengono ognuno una sequenza di numeri interi. Supponiamo che i nomi di questo due file siano `uno` e `due`. Si vuole fare in modo che il file risultato, di nome `ris`, contenga anch'esso una sequenza di interi, di cui il primo è ottenuto facendo la sottrazione fra il primo intero di `uno` e il primo intero di `due`. Il secondo intero deve essere ottenuto per sottrazione fra il secondo di `uno` e il secondo di `due`, ecc.

Questo esercizio si potrebbe risolvere usando i vettori, ossia leggendo tutti gli interi del primo file e mettendoli in un vettore, e lo stesso per il secondo file. Facendo la sottrazione elemento per elemento, e scrivendo il risultato su file, si ottiene il risultato voluto

Vediamo ora una soluzione che non usa vettore, ed è quindi più efficiente come occupazione di memoria. Usiamo questo algoritmo: dopo aver aperto i due file, leggiamo un intero dal primo file e uno dal secondo; facciamo la sottrazione e scriviamo il risultato sul terzo file.

Per implementare questo algoritmo, dobbiamo aprire tutti e tre i file, i primi due in lettura e il terzo in scrittura. Infatti, per poter accedere a un file, occorre prima aprirlo. Le successive operazioni di lettura e scrittura usano il valore del descrittore di file per capire su quale file occorre leggere o scrivere. Quindi, ci servono tre variabili per memorizzare i tre descrittori di file che si ottengono dall'apertura dei tre file.

Definiamo quindi le tre variabili:

```
  FILE *piu, *meno, *out;
```

Quando apriamo il primo file, assegnamo a `piu` il risultato della apertura del file `uno`, ossia il descrittore di questo file. Facendo lo stesso per gli altri file, le tre variabili `piu`, `meno` e `out` possono venire usate per dire alle funzioni `fscanf` e `fprintf` su quali file operare. Per esempio, per leggere un intero dal file `uno` passiamo a `fscanf` il descrittore del primo file, cioè `piu`:

```
  fscanf(piu, "%d", &x);
```

Il programma completo sottrai.c è riportato qui sotto. Quando uno dei due file di input (`piu` e `meno`) è terminato, si smette di leggere e si chiudono tutti e tre i file. Quindi, il terzo file contiene un numero di interi che è il minimo fra il numero di elementi del primo e del secondo file.

```
/*
  Sottrae, elemento per elemento, il contenuto
di due file
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *piu, *meno, *out;
  int x, y;
  int res;
```

```
                /* apre il primo file in lettura */
  piu=fopen("piu", "r");
  if( piu==NULL ) {
    perror("Errore in apertura del primo file");
    exit(1);
  }


                /* apre il secondo file in lettura */
  meno=fopen("meno", "r");
  if( meno==NULL ) {
    perror("Errore in apertura del secondo file");
    exit(1);
  }


                /* are il file in scrittura */
  out=fopen("ris", "w");
  if( out==NULL ) {
    perror("Errore in apertura del file risultato");
    exit(1);
  }


                /* legge i file */
  while(1) {
    res=fscanf(piu, "%d", &x);
    if( res!=1 )
      break;

    res=fscanf(meno, "%d", &y);
    if( res!=1 )
      break;

    fprintf(out, "%d\n", x-y);
  }


                /* chiude i file */
  fclose(piu);
  fclose(meno);
  fclose(out);

  return 0;
}
```

La cosa importante da notare in questo programma è che è possibile avere più file aperti contemporaneamente: in questo caso, i due file in lettura e il file in scrittura restano aperti contemporaneamente. Per dire alle funzioni `fscanf` e `fprintf` quale è il file da cui devono leggere o scrivere, si usa il descrittore di file.

Non in programma: Lettura di un file per righe

Lettura di un file per righe

La funzione `fscanf` permette di leggere tutti i dati di tipo scalare, ossia interi, caratteri e reale. Permette inoltre di leggere stringhe, utilizzando il formato `%s`. Leggere una stringa in questo modo può a volte portare a risultati che non ci si aspetta: la funzione `fscanf`, infatti, considera una stringa in input conclusa quando incontra uno spazio.

In alcuni casi può essere necessario leggere invece una intera riga da file. Per questa ragione, è stata introdotta la funzione `fgets`. L'uso tipico di questa funzione è quello di leggere una intera riga da un file di testo, e poi suddividere questa linea usando la funzione di lettura da stringhe `sscanf` che si vedrà più avanti. Questa funzione viene anche usata quando una linea in ingresso va considerata come una sola stringa, su cui poi si opera direttamente con le funzioni su stringhe.

La funzione `fgets` ha tre argomenti: il primo è un vettore di caratteri in cui va memorizzata la linea del file di testo; il secondo è il numero massimo di caratteri che si vogliono mettere in questa stringa; il terzo è un descrittore di file, e indica da quale file si vuole leggere la stringa. Il prototipo di questa funzione è il seguente:

```
char *fgets(char *s, int size, FILE *fd);
```

Diamo una descrizione degli argomenti e del valore di ritorno di questa funzione:

`char *s`
: questo è un puntatore a una zona di memoria in cui viene memorizzata la stringa letta; si noti che la zona di memoria deve già essere stata allocata, o staticamente oppure usando `malloc`; in altre parole la funzione `fgets` non alloca la memoria in cui mettere la stringa: passare alla funzione un puntatore non inizializzato è un errore;

`int size`
: il numero memorizzato in questa variabile dice alla funzione quale è la dimensione della zona di memoria puntata da `s`; in questo modo, la funzione sa che non può scrivere nella zona di memoria più di `size` caratteri, altrimenti finirà per scrivere dei valori in zone di memoria al di fuori di quella allocata per `s`; la funzione non scrive mai più di `size` caratteri, anche se la linea del file è più lunga;

`FILE *fd`
: questo è il descrittore di un file, e indica da quale file bisogna leggere

valore di ritorno
: coincide con il primo argomento della funzione (`s`) se è stato letto almeno un carattere (incluso il ritorno a capo); altrimenti ritorna `NULL`.

Capire quando il file è finito è semplice: basta infatti verificare se il valore risultato vale `NULL`. Se il risultato è `NULL`, allora non è stato possibile leggere neanche un carattere. In questo caso, la zona di memoria puntata da `s` non contiene un valore significativo, per cui non va elaborata.

Esistono ovviamente casi in cui viene letta una linea da file ‘‘ogni tanto’’, ma di solito la funzione `fgets` viene usata per leggere tutto il contenuto di un file riga per riga. Su ogni riga vengono poi fatte delle elaborazioni.

La struttura di un programma di questo genere è: prima si apre il file in lettura (con controllo errori), e poi si entra in un ciclo, in cui si legge una riga a ogni iterazione. In ogni iterazione, si legge una riga, e la si elabora. Se l'operazione di lettura ha dato risultato `NULL` allora vuol dire che non è stata letta nessuna riga, per cui `s` non va elaborata, e si deve invece uscire dal ciclo.

1. apri il file
2. controllo errore in apertura
3. ciclo:
    a) leggi riga
    b) se non è stata letta la riga, esci
    c) elabora le riga
4. chiudi il file

Il programma righe.c riportato qui sotto legge un file riga per riga. La elaborazione di una riga è in questo caso semplicemente le sua stampa su schermo. Si noti che la funzione `fgets` mette nel vettore `s` tutta la riga letta, *incluso il carattere di fine linea*. È per questo che la istruzione di stampa usa il formato `"%s"` invece che `"%s\n"`: infatti, il carattere di andata a capo si trova già nella stringa letta da file.

```
/*
  Lettura di un file riga per riga.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;
  char buf[200];
  char *res;


                /* apre il file */
  fd=fopen("righe.txt", "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* legge e stampa ogni riga */
  while(1) {
    res=fgets(buf, 200, fd);
    if( res==NULL )
      break;
    printf("%s", buf);
  }


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Si faccia attenzione al fatto che il valore di ritono della funzione `fgets` (che serve per capire quando il file è finito) è di tipo puntatore a carattere (ossia `char *`), e non intero come nel caso di `fscanf`. Questo puntatore non deve ovviamente venire inizializzato.

Linea più lunga di un file

Linea più lunga di un file

In questo esercizio facciamo uso della funzione di lettura di una riga da file. Sia dato un file di testo, di cui assumiamo che le linee siano tutte lunghe al più 200 caratteri. Si vuole determinare quale è la lunghezza della linea più lunga del file. Per determinare la lunghezza di una stringa, possiamo usare la funzione `strlen`, che prende una stringa come argomento e ritorna la sua lunghezza. Se si preferisce, si può anche risolvere l'esercizio scrivendo una funzione che calcola quanti elementi ci sono nella stringa prima del carattere di fine stringa `'\0'`.

L'esercizio consiste in una lettura di un file linea per linea. In questo caso, la elaborazione consiste nel confronto fra la lunghezza della linea e la massima lunghezza di linea trovata fino ad ora. Usiamo quindi una variabile intera `max` per indicare quale è la massima lunghezza di una linea trovata fino a questo momento. Per ogni linea letta, confrontiamo la sua lunghezza con il contenuto di questa variabile, eventualmente aggiornando il valore di `max`.

Il programma completo piulunga.c è riportato qui sotto.

```
/*
   Trova la massima lunghezza di linea
in un file di testo.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;
  char buf[200];
  char *res;
  int max;


                /* apre il file */
  fd=fopen("piulunga.txt", "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* lettura per righe */
  max=0;
  while(1) {
    res=fgets(buf, 200, fd);
    if( res==NULL )
      break;
    if( strlen(buf)-1 > max )
      max=strlen(buf)-1;
  }


                /* chiude il file */
  fclose(fd);


                /* stampa la massima lunghezza */
```

```
  printf("Lunghezza massima di linea: %d\n", max);

  return 0;
}
```

Si noti che la lunghezza della linea letta è `strlen(buf)-1` e non `strlen(buf)` come ci si potrebbe aspettare. Per capire come mai il programma funziona in questo modo, osserviamo che la funzione `fgets` legge una intera linea *incluso il carattere di fine linea*. Questo significa che se una linea è lunga dieci caratteri, la stringa che viene memorizzata è lunga undici caratteri, cioè i dieci che compongono effettivamente la linea più il carattere di ritorno a capo `'\n'`. La funzione `strlen` ritorna la lunghezza complessiva della stringa memorizzata, metre per noi la lunghezza di una linea di testo è il numero di caratteri nella linea, escluso quindi il carattere di ritorno a capo.

Se ci sono dubbi sul funzionamento della funzione `fgets`, si può provare a scrivere un programma che legge un file linea per linea, e per ogni linea stampa i valori numerici dei caratteri che stanno nella zona di memoria che contiene la linea letta. Questo dovrebbe chiarire cosa c'è nella zona di memoria dove viene messa le linea letta. Il programma leggilinea.c fa esattamente questo. Si provi ad eseguire su un file di testo con linee non più lunghe di diciannove caratteri, e si osservi il risultato. Il numero `10` è il valore numerico di `'\n'`, mentre `0` è il terminatore di stringa.

Copia di un file

Copia di un file

Questo esercizio consiste nel copiare, riga per riga, un file su un altro. Supponiamo che il nome del file di partenza sia `copia.txt` e il file destinazione si debba chiamare `x`.

La soluzione è abbastanza ovvia: si aprono i due file (il primo in lettura e il secondo in scrittura). Poi si legge dal primo file una linea per volta, e se questa operazione di lettura ha successo si scrive la linea sul secondo file, altrimenti si esce dal ciclo di lettura.

Il programma copia.c realizza questo algoritmo.

```
/*
  Copia un file riga per riga
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *in, *out;
  char buf[200];
  char *res;


                /* apre il file da leggere */
  in=fopen("copia.txt", "r");
  if( in==NULL ) {
    perror("Errore in apertura del file da leggere");
    exit(1);
  }


                /* apre il file da scrivere */
  out=fopen("x", "w");
```

```
  if( out==NULL ) {
    perror("Errore in apertura del file da scrivere");
    exit(1);
  }


                /* legge e scrive ogni riga */
  while(1) {
    res=fgets(buf, 200, in);
    if( res==NULL )
      break;
    fputs(buf, out);
  }


                /* chiude i file */
  fclose(in);
  fclose(out);

  return 0;
}
```

Per scrivere la linea sul secondo file si è usata la funzione `fputs`, che scrive una stringa su file. Al suo posto si sarebbe potuta usare la funzione `fprintf`, facendo `fprintf(fd, "%s", buf)`, e il risultato sarebbe stato identico.

Commenti in un file

Commenti in un file

Questo esercizio consiste nello scrivere un programma che stampa tutte le linee di un file che non sono commenti, dove un commento è semplicemente una linea che inizia con il carattere `'#'`.

Si tratta chiaramente di leggere un file linea per linea. Ogni volta che si legge una riga, si controlla se il primo carattere è `'#'`, e si stampa la linea su schermo solo in caso contrario.

Il programma completo commenti.c è riportato qui sotto.

```
/*
  Stampa le linee di un file che non sono commenti,
dove i commenti sono linee che iniziano con il
carattere #
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;
  char buf[200];
  char *res;


                /* apre il file */
  fd=fopen("commenti.txt", "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
```

```
  }


                /* lettura linea per linea */
  while(1) {
    res=fgets(buf, 200, fd);
    if(res==NULL)
      break;

    if( buf[0] != '#' )
      printf("%s", buf);
  }


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Si noti che non vengono considerati commenti le linee in cui il carattere `'#'` non è in prima posizione. Non è difficile modificare il programma in modo tale che sia considerato come commento anche solo una parte di linea che segue un carattere `'#'`. Il programma commentidue.c è la versione modificata del programma di sopra. Si noti che, in questo caso, una linea che inizia con `'#'` viene stampata come linea vuota.

Lettura da stringa

Lettura da stringa

La funzione `fgets` è molto utile quando un file di testo va elaborato una linea per volta. Esistono dei casi in cui non è da sola sufficiente. Consideriamo per esempio il caso in cui si vuole leggere da file degli interi, ignorando le linee che iniziano con un carattere `'#'`. In questo caso, per saltare le linee che iniziano con `'#'` la funzione `fgets` risulta comoda, mentre per leggere degli interi la funzione `fscanf` è la scelta migliore. Il problema è che, una volta letta una linea da file con `fgets`, se si cerca di fare una lettura con `fscanf` quello che si legge è il seguito del file, ossia la linea letta da `fgets` non viene poi scandita di nuovo da `fscanf`.

Una delle funzioni di libreria del C che risulta utile in questi casi è la funzione di lettura da stringa (esistono molte altre situazioni in cui questa funzione può essere utile). Questa funzione si chiama `sscanf`, e ha gli stessi argomenti della `fscanf` tranne il primo, che è una stringa invece che un descrittore di file. Questa funzione legge i dati dalla stringa invece che da file, ma per il resto il comportamento è identico.

Il seguente programma mediacommenti.c è un esempio di uso di questa funzione: si legge un file di testo, in cui le linee che iniziano con il carattere `'#'` sono commenti, mentre le altre linee contengono due numeri interi ciascuna.

```
/*
  Calcola la media degli interi su file.
Le linee che iniziano con # sono commenti
e non vengono considerate.
*/

#include<stdlib.h>
```

```
#include<stdio.h>

int main() {
  FILE *fd;
  char buf[200];
  char *res;
  int x, y;
  int n, somma, media;


                        /* apre il file */
  fd=fopen("mediacommenti.txt", "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                        /* legge una riga per volta */
  somma=0;
  n=0;
  while(1) {
    res=fgets(buf, 200, fd);
    if( res==NULL )
      break;

    if( buf[0]!='#' ) {
      sscanf(buf, "%d %d", &x, &y);
      somma+=(x+y);
      n+=2;
    }
  }


                        /* stampa la media */
  printf("La media e' %d\n", somma/n);

  return 0;
}
```

Un possibile miglioramento di questo programma è il controllo che le linee che non sono commenti contengono effettivamente due interi. Questo controllo si può fare tenendo conto che il valore di ritorno della funzione `sscanf` è lo stesso valore che avrebbe `fscanf` se la stringa fosse il contenuto di un file.

Stampa le righe con la data di oggi

Stampa le righe con la data di oggi

Si supponga di aver memorizzato degli appuntamenti su un file, in questo modo: ogni linea contiene una data, nel formato `giorno/mese/anno`, uno o più spazi, e una frase che descrive l'appuntamento.

Data una stringa che descrive una data, vogliamo sapere quali sono gli appuntamenti previsti per quella data. Quindi, se la stringa è `"2/3/2001"`, allora vogliamo stampare tutte le linee che iniziano con questa stringa.

Sono ovviamente possibili diverse soluzioni a questo problema. Sappiamo però che ò necessario leggere una riga di file per volta (dal momento che è questo che va stampato alla fine se la data coincide con quella data). Inoltre, è necessario confrontare la parte iniziale della linea con la data di oggi. Per confrontare due stringhe il C mette a disposizione la funzione `strcmp`, che restituisce `0` solo se le due stringhe passate come argomento sono uguali.

L'unica difficoltà, a questo punto, è quella di estrarre dalla linea letta da file la parte iniziale che descrive la data. Per fare questo, possiamo usare la funzione `sscanf`, facendo una lettura di stringa con il formato `"%s"`.

Il programma finale data.c legge una linea da file per volta. Per ogni linea, legge la stringa iniziale con `sscanf(buf, "%s", primo)`: in questo modo, `primo` contiene ora la prima parte della stringa `buf` (fino al primo spazio). Per verificare se la parte iniziale della linea coincide con la stringa data, si può ora usare la funzione `strcmp`. Se questa funzione ritorna `0`, si stampa l'intera linea.

```
/*
  Cerca in un file le righe che iniziano
con la data corrente. Usa una striga
costante per la data.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;
  char *oggi="2/3/2001";
  char buf[200];
  char *res;
  char primo[200];


                /* apre il file */
  fd=fopen("date.txt", "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* scansione del file */
  while(1) {

    res=fgets(buf, 200, fd);
    if( res==NULL )
      break;

    sscanf(buf, "%s", primo);

    if( !strcmp(primo,oggi) )
       printf("%s", buf);
  }


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Riportarsi all’inizio di un file

La funzione `rewind` permette di riposizionarsi all’inizio di un file. Nel caso di file aperti in lettura, è equivalente a chiudere il file ed aprirlo di nuovo. Il programma inizio.c riportato qui sotto fa vedere una applicazione di questa funzione: un file viene letto e stampato su schermo riga per riga. Quando si arriva alla fine, viene chiamata la funzione `rewind` che ‘‘riavvolge’’ il file, ossia ci riporta all’inizio del file: la successiva operazione di lettura legge l’inizio del file. Il programma effettua poi un secondo ciclo di lettura e stampa. L’effetto finale è quello di leggere e stampare due volte il file.

```
/*
  Stampa due volte un file.
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  FILE *fd;
  char buf[200];
  char *res;

                /* apre il file */
  fd=fopen("array.txt", "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


                /* legge e stampa fino all’eof */
  while(1) {
    res=fgets(buf, 200, fd);
    if( res==NULL )
      break;
    printf("%s", buf);
  }


                /* ricomincia dall’inizio */
  rewind(fd);


                /* legge e stampa fino all’eof */
  while(1) {
    res=fgets(buf, 200, fd);
    if( res==NULL )
      break;
    printf("%s", buf);
  }


                /* chiude il file */
  fclose(fd);

  return 0;
}
```

Cancellazione di un file

La cancellazione di un file si può fare con la funzione `remove`. Questa funzione prende un solo argomento, il nome del file da cancellare. Per esempio, il seguente programma cancella.c cancella il file di nome `"x"`. La funzione ritorna `0` se la cancellazione ha avuto successo, `-1` altrimenti (per esempio, se il file da cancellare non esiste).

```
/*
  Cancella un file
*/

#include<stdlib.h>
#include<stdio.h>

int main() {
  int res;

  res=remove("x");
  if( res!=0 )
    perror("Errore nella rimozione del file");

  return 0;
}
```