File binari

I file binari sono file in cui i dati sono memorizzati nello stesso modo in cui si trovano in memoria, per cui, per esempio, un intero occupa su un file binario sempre `sizeof(int)` byte, indipendentemente dal suo valore. Al contrario, in un file di testo il numero 0 occupa 1 byte mentre 92134 occupa cinque byte (ha cinque cifre, e quindi servono cinque caratteri per scriverlo). I vantaggi dei file binari rispetto a quelli di testo sono: minore dimensione (in media), facilità di modificare il file, facilità di riposizionarsi nel file. Gli svantaggi sono la non portabilità da un tipo di calcolatore a un altro, e il fatto che non si può creare o modificare un file binario usando un editor di testi.

Rappresentazione di dati come testo e in forma binaria

Rappresentazione di dati come testo e in forma binaria

Consideriamo una variabile di tipo intera `x`, e assumiamo che sul calcolatore servano quattro byte per rappresentarla, ossia che `sizeof(int)=4`. Come sono fatti i byte che si trovano in questa zona di memoria? È sufficiente considerare la rappresentazione binaria del numero, e considerare che ogni byte sono otto bit. Per esempio, per rappresentare il numero `943123`, passiamo alla sua rappresentazione binaria:

```
00000000000011100110010000010011
```

Dividendo questa sequenza di 32 bit in pezzi di otto bit ciasuno (byte) si ottiene:

```
00000000 00001110 01100100 00010011
```

Il valore di questi byte, scritto in decimale, è:

```
0 14 100 19
```

Cosa succede quando si va a scrivere il numero `943123` su un file di testo? L'istruzione `fprintf(fd, "%d", x);` scrive su file le cifre decimali del numero, ossia scrive la sequenza di byte che rappresentano i caratteri:

```
'9' '4' '1' '2' '3'
```

Stampando il valore numerico di questi byte, per esempio con `fprintf("%d", '9');` si ottengono i seguenti valori:

```
57 52 49 50 51
```

Infatti, il carattere `'9'` è rappresentato in codice ASCII da numero `57`, ecc.

Cosa si ricava da tutto questo discorso? Lo stesso numero, `943123` viene rappresentato in memoria con quattro byte (come tutti gli interi), i cui valori (0 14 100 19) dipendono dal metodo di rappresentazione in binario. Se si scrive il numero su un file di testo, questo viene rappresentato con la sequenza di byte `57 52 49 50 51`, che sono i valori numerici delle varie cifre del numero.

Il C permette di scrivere su un file dei valori esattamente nel modo in cui sono rappresentati in memoria. I file scritti in questo modo si dicono *file binari*, per contrapporli ai file di testo, in cui i dati vengono scritti usando la loro rappresentazione in forma si sequenza di caratteri.

Scrivere su un file binario

Per utilizzare un file binario, va per prima cosa aperto. Le funzioni di apertura e chiusura di un file binario sono esattamente le stesse dei file di testo. Del resto, i file binari differiscono da quelli di testo solo per il modo in cui le informazioni sono rappresentate al loro interno.

Per scrivere su un file binario, si deve prima aprire usando la funzione `fopen`, passando come modo la stringa `"w"`. Per scrivere un dato sul file, si usa la funzione `fwrite`. Questa funzione ha quattro argomenti: il primo è l'indirizzo del dato da scrivere, il secondo è la sua dimensione, il terzo per il momento vale 1, il quarto è il descrittore di file. Per esempio, se `x` è una variabile intera, allora la istruzione:

```
fwrite(&x, sizeof(x), 1, fd);
```

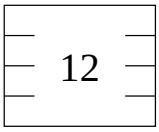
Scrive il valore della variabile `x` sul file, in modo binario. Si possono ovviamente scrivere su file i valori di variabili di tipo generico. Si noti che non è possibile scrivere su un file binario un valore costante. In altre parole, per scrivere il valore `12` in un file binario occorre prima memorizzarlo in una variabile, e poi scriverlo con l'istruzione di sopra.

Come per i file di testo, quando si scrive più volte su un file binario, i dati vengono scritti l'uno dietro l'altro. Per esempio, il programma riportato sotto scrivi.c scrive su file un numero in formato binario, poi il numero superiore di uno e il numero inferiore di uno.

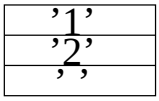
```
/*
  Scrive un intero su file binario, e poi
il numero superiore e inferiore.
*/

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

int main() {
  FILE *fd;
  int x=12;

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


                        /* scrive il numero */
  fwrite(&x, sizeof(int), 1, fd);


                        /* incrementa e scrive */
  x++;
  fwrite(&x, sizeof(int), 1, fd);



                        /* decrementa e scrive */
  x-=2;
```

```
  fwrite(&x, sizeof(int), 1, fd);


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

  return 0;
}
```

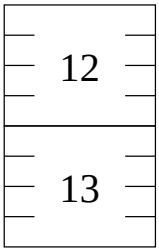
Confrontiamo ora il contenuto dei file, nel caso in cui questi tre numeri `12, 13, 11` sono scritti usando `fprintf` e nel caso in cui vengono scritti usando `fwrite`. Le figure qui sotto indicano quale è il contenuto del file nel caso di scrittura binaria (a sinistra) e di testo (a destra).



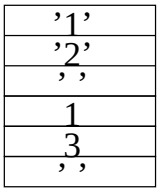
Quando si scrive un numero intero in formato binario, vengono scritti i byte che lo rappresentano. Nel nostro caso, assumiamo che bastano quattro byte per rappresentare un intero. Da notare che si scrivono comunque quattro byte, independentemente dal numero di cifre del numero.



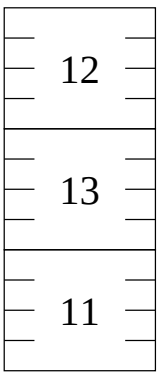
Quando si scrive il primo numero in un file di testo, vengono scritti i caratteri che corrispondono alle cifre del numero. Nel nostro caso, vengono scritti i due caratteri `'1'` e `'2'`. Per separare due interi su file di testo si scrive di solito uno spazio fra loro (oppure un carattere di ritorno a capo).



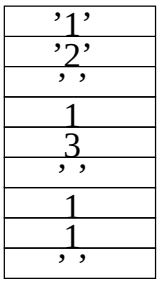
Viene fatto lo stesso per il secondo numero, e questi sono altri quattro byte che vengono scritti sul file.



Il secondo numero che viene scritto su file è 13, che occupa ancora due caratteri del file, più la spaziatura.



Alla fine, il file è composto da dodici byte. Ogni gruppo di quattro byte è ha i valori che stanno in memoria quando una variabile ha il valore del numero.



Il file è composto da nove byte. Ogni byte rappresenta il valore di un carattere che si usa per scrivere una cifra del numero.

Leggere da un file binario

Per leggere un dato da un file binario si usa la funzione `fread`, che ha gli stessi parametri della funzione `fwrite`. Il primo parametro è l'indirizzo della variabile su cui scrivere il dato, e gli altri sono gli stessi (il numero di byte in memoria occupati dal dato, il numero 1, e il descrittore di file). Per esempio, per leggere un intero da file e memorizzarlo nella variabile `x`, si può fare così:

```
fread(&x, sizeof(x), 1, fd);
```

Questo vale ovviamente per un qualsiasi tipo di dato. Il programma di esempio leggidue.c legge due interi da un file binario e li stampa su schermo.

```
/*
  Legge due interi da un file binario.
*/

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

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

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


                        /* lettura primo numero */
  res=fread(&x, sizeof(int), 1, fd);
  if( res!=1 ) {
    perror("Errore in lettura");
    exit(1);
  }


                        /* lettura secondo numero */
  res=fread(&y, sizeof(int), 1, fd);
  if( res!=1 ) {
    perror("Errore in lettura");
    exit(1);
  }


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


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

  return 0;
}
```

La funzione ritorna il valore 1 se la lettura è avvenuta correttamente, e 0 altrimenti (questa è una prima approssimazione, che va bene per come usiamo la funzione per ora).

Si noti che la funzione `fread`, se riesce a leggere almeno quattro byte da file, riporta che la lettura di un intero è avvenuta correttamente. Quindi non esiste un ‘‘errore di conversione’’ come nel caso in cui si legge da un file di testo e si incontra un carattere al posto di una cifra. Infatti, un gruppo di quattro byte corrisponde sempre a un intero. L’unico caso in cui la funzione `fread` non ritorna 1 è perchè non è stato possibile leggere un numero sufficente di byte per riempire il tipo di dato da leggere. Questo può essere dovuto a un problema hardware, di sistema operativo, oppure al fatto che il file è finito.

Lettura da file binario fino alla fine del file

Lettura da file binario fino alla fine del file

È possibile leggere da un file binario senza sapere esattamente quanti dati ci sono scritti. Questo si realizza esattamente come nel caso dei file di testo: si continua a leggere fino al punto in cui si raggiunge la fine del file e si a questo punto, evidentemente, si smette di leggere.

Nel caso dei file binari, quando si tenta di leggere qualcosa, e il file è finito, la funzione `fread` ritorna 0 invece che 1. In effetti, la funzione può ritornare 0 anche in caso di errore hardware o di sistema operativo, ma per il momento questo non ci interessa.

Possiamo quindi leggere un file con un ciclo da cui si esce non appena il valore di ritorno di `fread` è zero. Il programma seguente leggi.c legge interi da file, e li stampa. Si esce dal ciclo di lettura/stampa appena si arriva alla fine del file. In sostanza è identico all’analogo programma di lettura da file di testo, soltanto che al posto della funzione di lettura da file di testo `fscanf` si usa la funzione di lettura da file binari `fread`.

```
/*
  Legge interi da un file binario,
fino alla fine del file.
*/

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

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

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


                        /* ciclo di lettura */
  while(1) {
    res=fread(&x, sizeof(int), 1, fd);
    if( res!=1 )
      break;
```

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


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

  return 0;
}
```

Lettura di un array con una sola istruzione

Lettura di un array con una sola istruzione

La funzione `fread` permette di leggere un array da file con una sola istruzione. Per leggere un singolo intero si è visto che la istruzione necessaria è:

```
fread(puntatore a intero, sizeof(int), 1, fd);
```

Questa funzione in effetti legge una sequenza di dati, e li memorizza in ordine a partire dall'indirizzo passato come primo parametro. Il numero di dati da leggere viene dato come terzo argomento. Dato che fino ad ora abbiamo letto un solo intero, il numero che abbiamo passato alla funzione era 1. Nel caso in cui vogliamo leggere un certo numero di elementi, dobbiamo semplicemente passare questo numero come terzo parametro alla funzione.

Dal momento che un array è una zona di memoria in cui possiamo mettere dati di un certo tipo in modo consecutivo, per leggere `n` elementi e metterli in un array `vett`, dobbiamo semplicemente dire alla funzione di leggere `n` dati (quindi passiamo `n` come terzo argomento alla funzione), e di metterli nella zona di memoria che comincia dall'indirizzo `&vett`, e quindi passiamo `&vett` come indirizzo alla funzione `fread`.

Qui sotto riportiamo il programma leggiarray.c che legge un array di dieci elementi da file con una sola istruzione. Facciamo notare che il valore di ritorno della funzione `fread` è in effetti il numero di elementi che la funzione è riuscita a leggere da file. Il programma può essere migliorato aggiungendo un controllo su questo valore di ritorno.

```
/*
  Legge un array da un file binario
(al massimo dieci elementi)
*/

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

int main() {
  FILE *fd;
  int vett[10];
  int n;
  int i;
  int res;

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

```
                        /* lettura vettore */
  n=fread(&vett, sizeof(int), 10, fd);


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


                        /* stampa il vettore */
  for(i=0; i<n; i++)
    printf("vett[%d]=%d\n", i, vett[i]);

  return 0;
}
```

Scrittura di interi casuali

Scrittura di interi casuali

Il seguente programma random.c usa le funzioni di libreria `srandom` e `random` per generare dei numeri casuali, per produrre un file binario che contiene degli interi casuali. L'uso di queste due funzioni non ci interessa. Questo programma può essere utile per testare dei programmi che leggono file binari, dal momento che questo genere di file non si può creare con un editor di testi.

Nel caso in cui serve testare un programma che legge dati di tipo diverso da intero, occorre chiaramente modificare questo programma in modo che scriva i dati di tipo opportuno.

```
/*
  Scrive interi random su un file binario.
*/

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

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

                        /* inizializza il generatore
                        di numeri casuali */
  srandom(time(NULL));


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


                        /* scrive i numeri */
  n=random()%20;
  for(i=0; i<=n-1; i++) {
    x=random()%30;
```

```
    fwrite(&x, sizeof(int), 1, fd);
  }


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

  return 0;
}
```

Apertura in lettura e scrittura

Apertura in lettura e scrittura

Fino ad ora abbiamo visto che è possibile aprire il file in lettura, oppure in scrittura. È in effetti possibile aprire i file in modo tale che sia possibile sia leggere che scrivere sul file.

Per aprire il file in modo che si possa sia leggere che scrivere, si passa come stringa di modalità alla funzione `fopen` la stringa `"r+"`. Per leggere si usa come al solito la funzione `fread` mentre per scrivere si usa la funzione `fwritef`.

Leggere e scrivere richiede però di introdurre il concetto di posizione corrente all'interno del file. Si consideri quindi il seguente programma dispari.c. Cosa succede se si prova ad eseguirlo su un file binario, visualizzando il contenuto del file prima e dopo? (per visualizzare il contenuto di un file binario, usare leggi.c).

```
/*
  Legge il primo numero, lo raddoppia, e lo
copia sul secondo. Fa lo stesso sul terzo e
sul quarto, ecc.
*/

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

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


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


                        /* ciclo di lettura */
  while(1) {

                        /* legge un intero */
    res=fread(&x, sizeof(int), 1, fd);
    if( res!=1 )
      break;
```

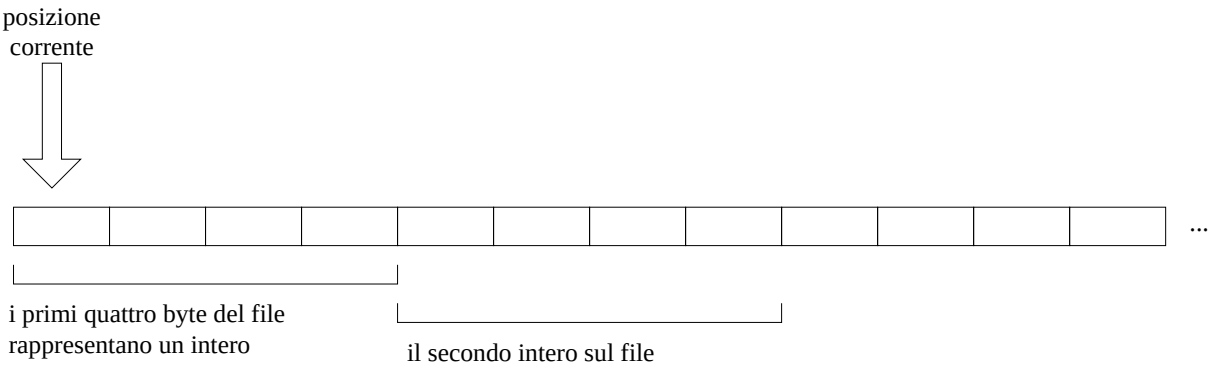
```
                        /* mette il doppio in y e lo scrive */
    y=2*x;
    fwrite(&y, sizeof(int), 1, fd);
  }


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

  return 0;
}
```

A prima vista, può sembrare che questo programma raddoppia tutti i numeri del file. Quello che succede, invece, è che tutti i numeri in posizione pari (il secondo, il quarto, ecc) sono stati sostituiti dal doppio del numero che li precede (per esempio, nella seconda posizione ora c'è il doppio del primo numero).

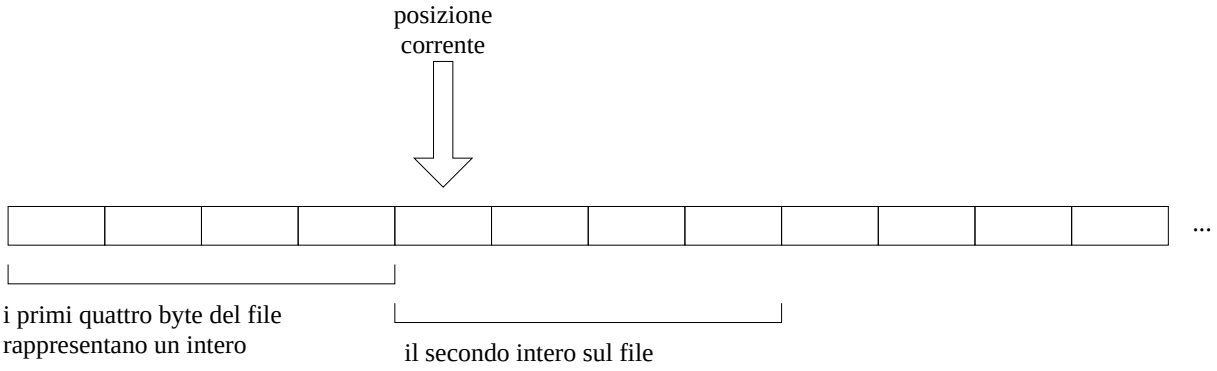
Per analizzare il motivo di questo comportamento, occorre introdurre un concetto nuovo, che è quello di *posizione all'interno del file*. Come si è notato parlando della differenza fra file binari e testuali, un file è in effetti semplicemente una sequenza di byte. Quello che li rende differenti da un vettore di caratteri è il fatto che il dispositivo in cui è memorizzato è il disco invece che la memoria primaria. Per il resto, hanno tutte le caratteristiche degli array.

Quelle che i file hanno in più rispetto agli array è il fatto di avere una posizione corrente di lettura definita in modo automatico. Questa posizione corrente è semplicemente un intero che indica la posizione nella sequenza in cui si va a leggere e scrivere. In particolare, quando si apre il file in lettura, la posizione corrente è l'inizio della sequenza. Ogni volta che si legge qualcosa, la posizione avanza.

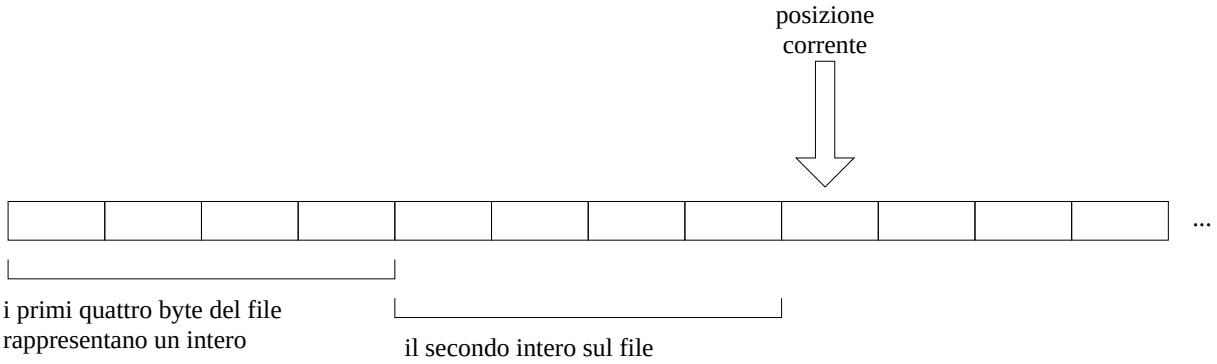
Quello che segue è una descrizione dettagliata delle operazioni di lettura e scrittura da file, e *valgono anche nei casi in cui il file viene aperto in sola lettura o sola scrittura*. Possiamo rappresentare un file come una sequenza di byte. All'apertura la posizione corrente indica il primo byte del file. Possiamo rappresentare questa cosa graficamente come segue:



L'effetto della operazione di lettura `fread(&x, sizeof(int), 1, fd)` è quello di leggere i primo quattro byte del file e metterli in una variabile intera (il numero di byte letti dipende ovviamente dal tipo della variabile). Questa operazione di lettura ha anche l'effetto di spostare la posizione corrente all'interno del file in avanti, al primo byte che segue quelli letti:



La lettura del secondo elemento ha ancora come effetto quello di spostare la posizione corrente in avanti.



Per essere precisi, possiamo dire che l'operazione di lettura si comporta come segue:

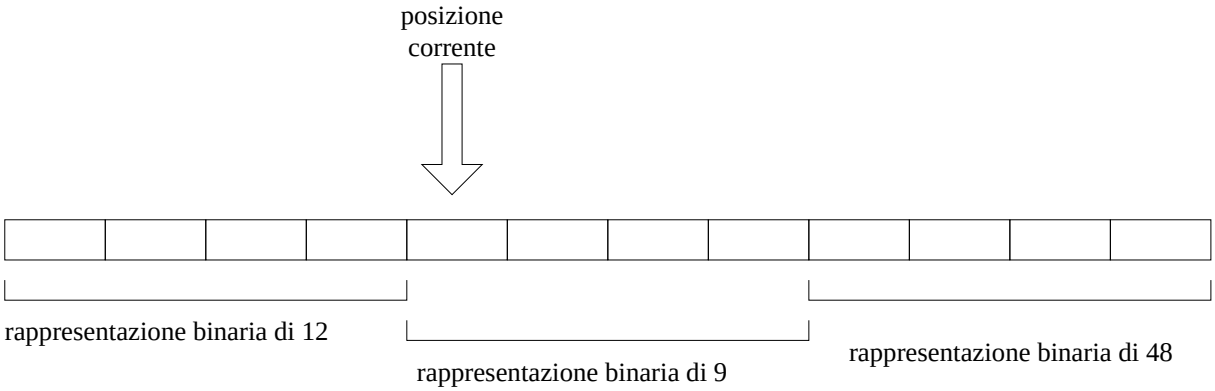
1. legge il dato che si trova a partire dalla posizione corrente
2. sposta la posizione corrente in avanti, al primo byte che segue quelli letti

Questo descrive cosa fa esattamente l'operazione di lettura. Il motivo per cui due successive operazioni di lettura leggono i dati in sequenza è che la prima, dopo aver letto, sposta la posizione in avanti, per cui la seconda operazione legge il secondo dato e non legge di nuovo il primo.

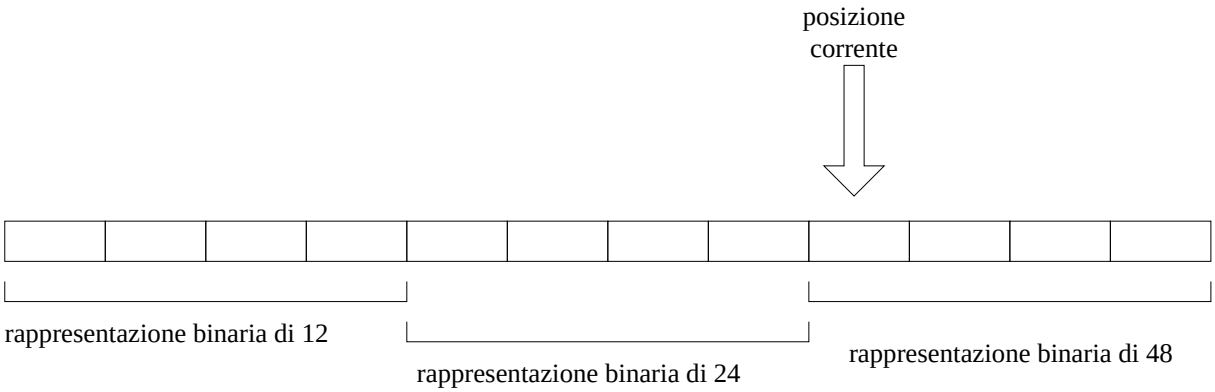
La stessa cosa succede quando si scrive: dopo aver scritto qualcosa, la posizione corrente avanza ancora. In altre parole, se si apre un file in scrittura, la posizione corrente è all'inizio del file. Quando si scrive, si scrive a partire dalla posizione corrente, e la posizione viene spostata nello stesso modo della lettura. Questo è il motivo per cui, quando si fanno più operazioni di scrittura su file, le cose vengono scritte in sequenza: in realtà, la funzione `fwrite` scrive sempre a partire dalla posizione corrente, solo che avanza questa posizione dopo aver scritto. Questo fa sí che la successiva operazione di scrittura scriva di seguito invece di sovrascrivere il dato scritto prima.

Vediamo ora cosa succede quando si esegue il programma di sopra. Supponiamo che nel file siano scritti, in forma binaria, i tre numeri `12`, `9` e `28`. Quando si apre, la posizione corrente è all'inizio del file:

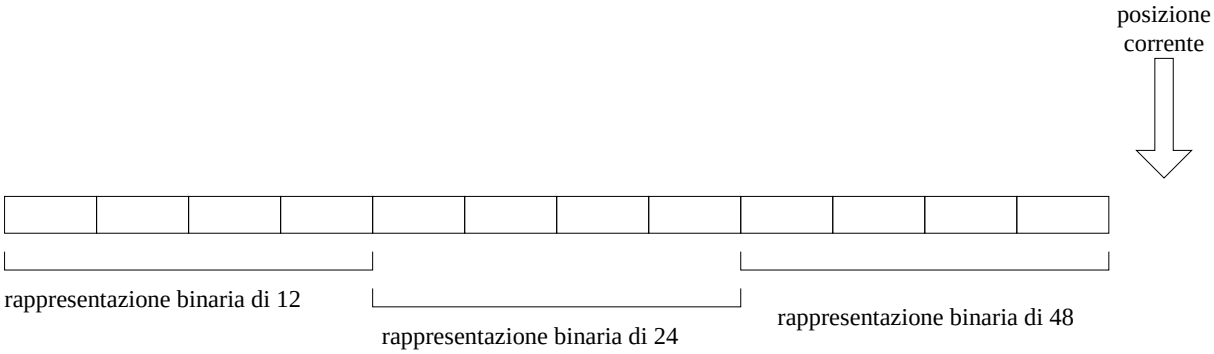
Si legge il primo intero a partire dalla posizione corrente, e si sposta in avanti la posizione. Ora x vale quanto letto da file, cioè 12.



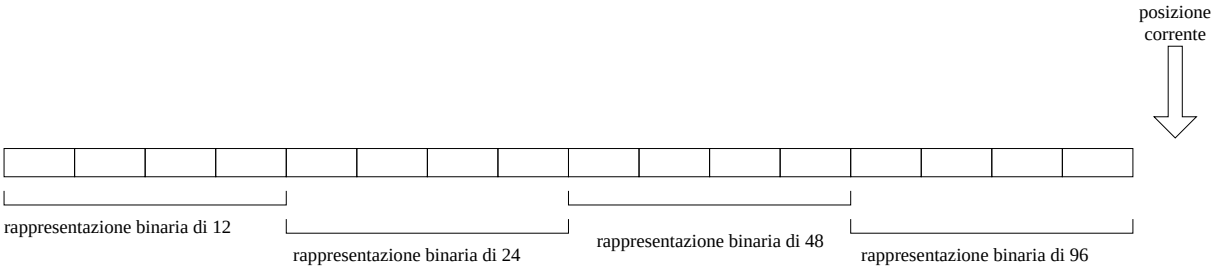
Si assegna a y il doppio del valore di x, cioè 24. Questo numero viene scritto in forma binaria nel file. La scrittura avviene a partire dalla posizione corrente, e sposta in avanti la posizione corrente. In questo momento, la posizione corrente è sopra il quinto byte del file, ed è a partire da questa posizione che si va a scrivere. La posizione avanza ancora di quattro, dato che si sta scrivendo un intero.



Ora si legge un intero, a partire dalla posizione corrente, per cui si legge 48. Si sposta la posizione corrente, che ora è sulla fine del file.



Il numero che si va a scrivere sul file è il doppio di quello letto, per cui si scrive 96. Questo numero viene scritto sotto la posizione corrente del file, e quindi si aggiungono quattro byte in coda al file. Lo stato del file alla fine è il seguente:



A questo punto si cerca di leggere di nuovo, ma l'operazione fallisce perchè il file è finito.

L'effetto che ne risulta è che il programma legge un numero, poi lo raddoppia e poi lo scrive. Però la scrittura avviene sulla posizione corrente dopo la lettura, per cui si scrive il doppio sulla posizione successiva. Quindi, per esempio, viene letto il primo numero e il doppio viene scritto sul secondo.

Riassumendo, i file si possono vedere come array, in cui la posizione corrente avanza a ogni lettura e scrittura. Nelle prossime pagine vediamo come si può modificare la posizione corrente nel file, con opportune istruzioni.

Osservazione: il concetto di posizione del file, e il modo in cui la si può modificare, vale per tutti i file, anche quelli di testo (del resto, la distinzione fra file di testo e binari dipende solo dal modo in cui viene interpretato il contenuto). Soltanto che per i file di testo modificare la posizione corrente è di minore utilità a causa della dimensione variabile dei dati numerici quando sono scritti come stringhe.

Come muoversi in un file binario

Come muoversi in un file binario

Le operazioni di lettura e scrittura hanno come effetto quello di spostare la posizione corrente nel file. È però anche possibile spostare la posizione corrente in modo arbitrario, senza leggere e scrivere. La funzione che si usa si chiama `fseek`.

Vediamo ora come si può portare avanti e indietro la posizione corrente. Per avanzare di `n` byte rispetto alla posizione corrente, si fa:

```
fseek(fd, n, SEEK_CUR);
```

Per tornare indietro di n byte, ossia spostare la posizione corrente di n byte indietro, si fa:

```
fseek(fd, -n, SEEK_CUR);
```

In altre parole, il secondo parametro indica la nuova posizione relativamente a quella vecchia.

Modifichiamo il programma della pagina precedente dispari.c, in modo tale che legga, uno per volta, tutti gli interi che stanno scritti su file, e al loro posto ci scriva il valore doppio. Questo si può ottenere in questo modo: una volta letto il numero, sappiamo che la posizione corrente è andata `sizeof(int)` posizioni avanti; per poter scrivere sopra al numero letto occorre quindi tornare indietro di `sizeof(int)` posizioni. Questo si realizza con l'istruzione

```
    fseek(fd, -sizeof(int), SEEK_CUR);
```

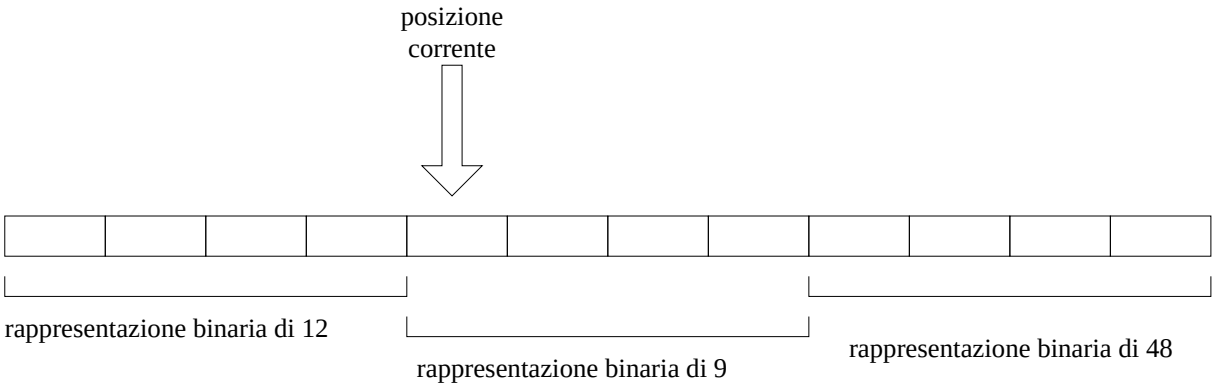
Quello che segue è il programma modificato raddoppia.c

```
/*
  Raddoppia tutti i numeri di un file.
*/

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

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


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


                        /* ciclo di lettura */
  while(1) {

                        /* legge un intero */
    res=fread(&x, sizeof(int), 1, fd);
    if( res!=1 )
      break;

                        /* riposiziona */
    fseek(fd, -sizeof(int), SEEK_CUR);

                        /* mette il doppio in y e lo scrive */
    y=2*x;
    fwrite(&y, sizeof(int), 1, fd);
  }


                        /* chiude il file */
```

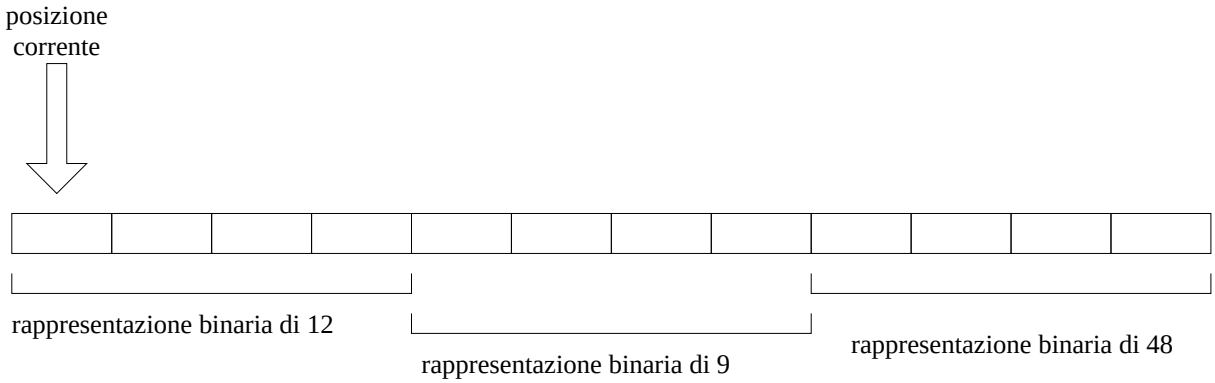
```
  fclose(fd);

  return 0;
}
```

Vediamo ora come funziona questo programma sul file di esempio. Quando il file viene aperto, la posizione corrente si trova sull'inizio del file:

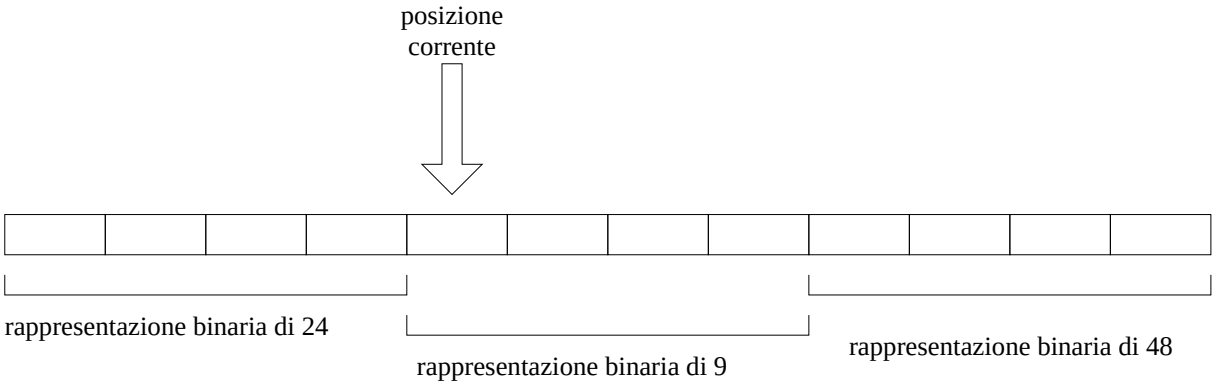
Si va ora a leggere il primo intero, che è 12. Questo valore viene memorizzato nella variabile `x`. L'istruzione di lettura ha anche l'effetto secondario di spostare in avanti la posizione corrente all'interno del file:



A questo punto, abbiamo l'istruzione `fseek(fd, -sizeof(int), SEEK_CUR);`, che sposta la posizione nel file indietro del numero di byte usati per rappresentare un intero. Dal momento che abbiamo prima letto un intero, e poi spostato indietro di un numero uguale di posizioni, quello che si ottiene alla fine è che la posizione nel file coincide con quella iniziale:



Si assegna ora ad `y` il doppio del valore di `x`, e si scrive `y` sul file. Si noti che il numero viene scritto *a partire dalla posizione corrente*. Quindi, viene scritto 24 (il doppio di 12) a partire dalla posizione iniziale del file. L'operazione di scrittura ha anche l'effetto di spostare in avanti la posizione corrente nel file:



L'effetto complessivo di leggere-spostare indietro-scrivere è che il numero letto viene sostituito con un altro (in questo caso, il doppio). Eseguendo un'altra iterazione del ciclo, si fa la lettura del secondo intero su file, si ritorna indietro, e si scrive il doppio nelle stesse posizioni in cui prima si trovava il secondo numero. Lo stesso vale anche per il terzo, ecc. Quindi, il programma effettivamente sostituisce a ogni numero il suo valore raddoppiato.

Nota: L'istruzione di spostamento della posizione corrente funziona anche sui file di testo. Su questi file è però molto meno utile, dal momento che lo spazio occupato dai dati non è costante. Se si volesse realizzare un programma analogo a quello di sopra, ma che operi su un file di testo, si dovrebbe tenere conto che, se si legge per esempio il numero 8, allora il doppio è 16, che occupa due byte (mentre il precedente ne occupa uno solo).

Scambio elementi a coppie

Scambio elementi a coppie

Risolviamo il seguente esercizio: dato un file binario che contiene una sequenza di numeri interi, scambiare il primo intero con il secondo, il terzo con il quarto, ecc.

L'esercizio si risolve semplicemente leggendo una coppia di numeri interi per volta, riportandosi indietro nel file fino alla posizione in cui si trovava il primo intero, e stampando i due interi a partire da questa posizione in ordine inverso. Facciamo quindi un ciclo in cui, dopo aver letto due interi `x` e `y`,

ci riportiamo indietro fino alla posizione iniziale del primo intero:

```
    fseek(fd, -2*sizeof(int), SEEK_CUR);
```

A questo punto possiamo scrivere i due interi su file. Dato che vanno invertiti, scriviamo primo il numero che è stato letto per secondo, cioè `y`. Il primo numero che è stato letto, lo scriviamo di seguito.

```
    fwrite(&y, sizeof(int), 1, fd);
    fwrite(&x, sizeof(int), 1, fd);
```

Il programma completo scambia.c è riportato qui sotto.

```
/*
  Scambia gli elementi di un file
a coppie (il primo con il secondo,
il terzo con il quarto, ecc.)
*/

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

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


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


                        /* ciclo di lettura */
  while(1) {
                        /* legge il primo */
    res=fread(&x, sizeof(int), 1, fd);
    if( res!=1 )
      break;

                        /* legge il secondo */
    res=fread(&y, sizeof(int), 1, fd);
    if( res!=1 )
      break;

                        /* riposiziona indietro */
    fseek(fd, -2*sizeof(int), SEEK_CUR);

                        /* scrive il secondo e poi il primo */
    fwrite(&y, sizeof(int), 1, fd);
    fwrite(&x, sizeof(int), 1, fd);
  }


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

  return 0;
}
```

Ordinamento a bolle

Vediamo ora come si può implementare l'algoritmo di ordinamento a bolle direttamente su file. Implementare questo algoritmo direttamente su file, invece di leggere tutto il file in un vettore, è necessario quando la dimensione del file è tale da non entrare tutto in memoria principale.

Usiamo qui una versione semplificata. Ogni passo consiste nella scansione, ed eventuale scambio, di tutti gli elementi del file. Si ripete questo passo fino a quando gli elementi non risultano ordinati.

La differenza fra questo programma e quello analogo di ordinamento su array è minima: a parte l'apertura e chiusura del file, l'unica cosa che occorre sapere è che per guardare quale valore ha un certo intero si usa le funzione `fread`, e che dopo aver letto, se si vuole scrivere al posto di quello che si è letto, occorre riposizionarsi nel file usando la funzione `fseek`.

Nel dettaglio:

1. leggiamo due interi
2. se non sono nell'ordine giusto:
    a) si torna indietro nel file di due interi
    b) si scrivono i numeri scambiati
3. si ritorna indietro di un intero nel file

Si noti che tornare indietro di un intero è necessario in entrambi i casi: se non ci sono stati scambi, siamo ora posizionati sul numero che segue l'ultimo letto, mentre ancora ci manca da fare il confronto fra l'ultimo letto e il successivo. Nel caso in cui c'è stato uno scambio, la posizione dopo la scrittura è ancora alla posizione dopo la coppia, per cui va spostata indietro per lo stesso motivo.

Il programma completo bubblesort.c è qui sotto.

```
/*
  Ordinamento a bolle
*/

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

int main() {
  FILE *fd;
  int res;
  int pos;

  int x[2];
  int ordinato;
  int temp;


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

```
                        /* scansione */
  ordinato=0;
  while( !ordinato ) {
    ordinato=1;

    while(1) {
      res=fread(x, sizeof(int), 2, fd); /* legge due interi */
      if( res!=2 )
        break;

      if( x[0]>x[1] ) {                 /* confronto e scambio */
        ordinato=0;

        temp=x[0];
        x[0]=x[1];
        x[1]=temp;

                                        /* riposiziona il file */
        fseek(fd, -2*sizeof(int), SEEK_CUR);

                                        /* scrive */
        res=fwrite(x, sizeof(int), 2, fd);
        if( res!=2 ) {
          perror("Errore in scrittura");
          exit(1);
        }
      }

      fseek(fd, -sizeof(int), SEEK_CUR);        /* torna indietro di uno */
    }

    rewind(fd);                 /* ricomincia dall'inizio */
  }


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

  return 0;
}
```

Trovare e usare la posizione assoluta all'intero di un file

Trovare e usare la posizione assoluta all'intero di un file

Fino ad ora abbiamo usato la funzione `fseek` per spostare in avanti e indietro la posizione del file, ma si trattava sempre di spostamenti relativi alla posizione corrente. La funzione `fseek` può anche venire usata per specificare la posizione assoluta all'interno del file. È per esempio possibile dire alla funzione che vogliamo che la posizione corrente sia sul decimo byte del file, per esempio.

Per dire che la posizione che vogliamo è assoluta invece che relativa, alla funzione `fseek` passiamo `SEEK_SET` come terzo argomento. In alternativa, possiamo dire che in questo modo stiamo dando la posizione relativa alla posizione iniziale. Per esempio:

```
fseek(fd, 0, SEEK_SET);
```

È equivalente a `rewind`, ossia ci si sposta all'inizio del file. Nello stesso modo,

```
fseek(fd, 10, SEEK_SET);
```

Sposta la posizione corrente di lettura e scrittura al decimo byte del file.

È anche possibile *determinare* la posizione corrente nel file. Si usa la funzione `ftell`, che restituisce un intero, che è appunto la posizione corrente. Se per esempio, questa funzione restituisce 20, allora vuol dire che la posizione corrente è sul ventesimo byte del file.

Il seguente programma risolve il seguente problema: dato un file di interi memorizzati in forma binaria, trovare il massimo, e scrive zero al suo posto.

Il programma cancellamax.c memorizza la posizione nel file in cui si è incontrato l'elemento più grande, si sposta in quella posizione e lo mette a 0 (cioè scrive 0 sopra quel numero). L'unica cosa che può essere poco chiara è il perchè facciamo `posmax=ftell(fd)-sizeof(int)`, ossia perchè togliamo `sizeof(int)` alla posizione corrente del file. Il motivo è semplicemente che la posizione corrente si trova sempre dopo quella in cui il numero `x` è stato letto. Se `x` è maggiore del massimo, allora la nuova posizione del massimo (cioè la posizione in cui è stato letto `x`) è di `sizeof(int)` indietro rispetto alla posizione corrente.

```
/*
  Mette a 0 il massimo elemento di un vettore.
*/

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

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

  int x;
  int max;              /* massimo trovato finora */
  int posmax;           /* posizione del massimo nel file */


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


                        /* ciclo di lettura */
  max=0;
  while(1) {
    res=fread(&x, sizeof(int), 1, fd);
    if( res!=1 )
      break;

    if( x>max ) {
      max=x;
      posmax=ftell(fd)-sizeof(int);
    }
  }
```

```
                        /* torna nella posizione del massimo
                           e ci scrive sopra 0 */
  fseek(fd, posmax, SEEK_SET);
  x=0;
  fwrite(&x, sizeof(int), 1, fd);


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

  return 0;
}
```

Sostituzione di stringhe

Sostituzione di stringhe

Esercizio: scrivere un programma che prende tre argomenti, i cui primi due sono stringhe e il terzo e il nome di un file. Questo programma deve controllare se nel file appare la prima stringa. Se appare, al suo posto deve mettere la seconda stringa. Occorre controllare che le due stringhe siano uguali.

L'esercizio si risolve in questo modo: si legge una stringa da file (ossia un array di caratteri) e lo si confronta con quello da sostituire. Se sono uguali, si riporta la posizione all'interno del file al punto in cui inizia la stringa letta, e si scrive la nuova stringa. Si passa poi a leggere la stringa che inizia alla posizione successiva del file.

Il ciclo di lettura delle stringhe si può riassumere nel seguente modo:

```
while(1) {
  memorizza posizione;
  leggi stringa;
  se non ci sono abbastanza caratteri su file, esci dal ciclo

  confronta stringa con la stringa da cercare;
  passa alla posizione +1;
}
```

Il punto fondamentale è che occorre controllare sia la stringa che parte dalla posizione 0 che quella che parte dalla posizione 1, 2, ecc. Se ci limitiamo a leggere senza riposizionare, avremmo solo la stringa che parte da 0, poi quella che segue, ecc. Per esempio, se la stringa da cercare è lunga 10 caratteri, avremmo fatto il controllo solo sulla stringa che parte dalla posizione 0, poi dalla posizione 10, poi 20, ecc, e quindi non ci saremmo accorti se la stringa era presente ma partiva per esempio dalla posizione 4.

Una volta letta la stringa, possiamo usare la funzione `memcmp`. Questa funzione verifica che due array siano uguali fino all'`n`-esimo elemento, dove `n` è il terzo parametro della funzione. Nel caso in cui le due zone siano uguali, ritorniamo alla posizione nel file prima della lettura, e scriviamo la nuova stringa.

Per controllare il corretto funzionamento del programma subst.c si può verificare cosa succede quando la stringa non viene trovata, oppure viene trovato solo un frammento iniziale, oppure viene trovata tutta, e quindi sostituita.

```
/*
  Sostituisce una stringa con un'altra,
in un file binario.
*/

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

int main(int argn, char *argv[]) {
  FILE *fd;
  int res;
  char *letto;
  int len;
  int i, j;
  int trovato;
  int pos, temp;


               /* controllo argomenti */
  if( argn-1!=3 ) {
    printf("Uso: subst stringa stringa file\n");
    printf("Sostituisce la prima stringa con la seconda\n");
    exit(1);
  }

  if( strlen(argv[1]) != strlen(argv[2]) ) {
    printf("Le due stringhe devono avere la stessa lunghezza\n");
    exit(1);
  }

  len=strlen(argv[1]);

  letto=malloc(len);

  printf("Sostituzione %s --> %s in %s\n", argv[1], argv[2], argv[3]);


               /* apre il file */
  fd=fopen(argv[3], "r+");
  if( fd==NULL ) {
    perror("Opening file");
    exit(1);
  }


               /* ciclo di lettura */
  while(1) {

               /* memorizza la posizione */
    pos=ftell(fd);


               /* legge una stringa */
    res=fread(letto, sizeof(char), len, fd);
    if( res<len )
      break;


               /* confronto ed eventuale sostituzione */
    if( ! memcmp(argv[1], letto, len) ) {
      fseek(fd, pos, SEEK_SET);
      fwrite(argv[2], sizeof(char), len, fd);
```

```
    }

                /* posizione avanzata di uno */
    fseek(fd, pos+1, SEEK_SET);
  }

                /* chiude il file */
  fclose(fd);
  return 0;
}
```

Dimensione di un file

Dimensione di un file

Un modo possibile per determinare la dimensione di un file è quello di aprire il file, posizionarsi alla fine del file, e usare `ftell` per vedere in che posizione ci troviamo.

Per posizionarsi alla fine del file usiamo l'istruzione `fseek(fd, 0, SEEK_END)`. Il terzo valore passato, ossia `SEEK_END`, dice che occorre fare un posizionamento non rispetto alla posizione corrente oppure all'inizio del file, ma rispetto alla fine del file. I tre valori che è possibile dare come terzo argomento alla funzione `fseek` hanno quindi il seguente significato:

`feeks(fd, n, SEEK_CUR)`
: sposta la posizione nel file di `n` posizioni rispetto alla posizione corrente;

`feeks(fd, n, SEEK_SET)`
: sposta la posizione nel file di `n` posizioni rispetto alla posizione iniziale, ossia la nuova posizione è esattamente `n`;

`feeks(fd, n, SEEK_END)`
: sposta la posizione nel file di `n` posizioni rispetto alla posizione finale del file.

Si ricordi che `n` può essere positivo oppure negativo. Un valore positivo indica uno spostamento in avanti, un valore negativo indica uno spostamento all'indietro, sempre rispetto alla posizione specificata come terzo argomento. Per esempio, per posizionarsi sul terz'ultimo byte di un file, occorre tenere presente che questo è uno spostamento alla fine del file *meno tre posizioni*, per cui occorre fare `fseek(fd, -3, SEEK_END)`.

Per posizionarsi alla fine del file, occorre quindi eseguire `fseek(fd, 0, SEEK_END);`. Si tenga presente che questa istruzione mette la posizione corrente al primo byte `dopo` l'ultimo byte del file. In altre parole, se dopo questa istruzione si scrive, il dato non va a sovrapporsi all'ultimo byte del file, ma viene scritto immediatamente dopo.

Dopo aver fatto il posizionamento alla fine del file, la funzione `ftell` dice qual'è la posizione nel file, e quindi determina il numero di byte del file (si consideri che la posizione parte da 0). Nel caso in cui il file contenga degli interi, si piò determinare quanti interi ci sono sul file dividendo il numero di byte per `sizeof(int)`.

dimensione.c

```
/*
  Determina il numero di byte di un file binario
(e il numero di interi che contiene).
*/

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

int main(int argn, char *argv[]) {
  FILE *fd;
  int size, dim;


                /* controllo argomenti */
  if(argn-1!=1) {
    printf("Errato numero di argomenti\n");
    exit(1);
  }

                /* apre il file */
  fd=fopen(argv[1], "r+");
  if(fd==NULL) {
    perror("Errore in apertura del file");
    exit(1);
  }

                /* determina la dimensione del file */
  fseek(fd, 0, SEEK_END);
  size=ftell(fd);
  printf("Numero di byte su file: %d\n", size);


                /* se il file contiene interi, dice quanti sono */
  dim=size/sizeof(int);
  printf("Numero di interi su file: %u/%d = %u\n", size, sizeof(int), dim);


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

  return 0;
}
```

Un secondo metodo per determinare la dimensione di un file consiste nell'uso della funzione `stat`. Questa funzione ha due parametri: il primo è il nome di un file (che non deve necessariamente essere stato aperto), e l'indirizzo di una variabile di tipo `struct stat`. Questa funzione modifica la struttura mettendoci i valori di alcune caratteristiche del file. In particolare, il campo `st_size` della struttura contiene la dimensione del file, ossia il numero di byte che contiene.

Per usare la funzione `stat` è necessaria la definizione della struttura `struct stat`, che si trova nel file header `sys/stat.h`. Il programma completo è dimstat.c che è riportato qui sotto.

```
/*
  Determina il numero di byte di un file binario
(e il numero di interi che contiene). Usa la
funzione stat
*/

#include<stdlib.h>
```

```
#include<stdio.h>
#include <sys/stat.h>

int main(int argn, char *argv[]) {
  int res;
  struct stat sstr;
  int size, dim;


                /* controllo argomenti */
  if(argn-1!=1) {
    printf("Errato numero di argomenti\n");
    exit(1);
  }

                /* apre il file */
  res=stat(argv[1], &sstr);
  if(res==-1) {
    perror("Errore in lettura del file");
    exit(1);
  }

                /* determina la dimensione del file */
  size=sstr.st_size;
  printf("Numero di byte su file: %d\n", size);


                /* se il file contiene interi, dice quanti sono */
  dim=size/sizeof(int);
  printf("Numero di interi su file: %u/%d = %u\n", size, sizeof(int), dim);


  return 0;
}
```

Ordinamento per selezione su file binario

Ordinamento per selezione su file binario

L’implementazione del metodo di ordinamento per selezione su file non presenta particolari difficoltà. L’unica differenza, rispetto al caso di ordinamento di un vettore, è che per accedere a un elemento occorre posizionarsi nel punto opportuno del file e leggere/scrivere il valore. Nel caso di ordinamento di numeri interi, l’accesso all’i-esimo elemento si ottiene facendo `fseek(fd, i*sizeof(int), fd)`, e poi leggendo/scrivendo il valore.

Dal momento che l’algoritmo confronta ogni volta un elemento i-esimo con tutti i successivi, si è scelto di leggere questo valore `x` da file una volta sola, e poi di confrontarlo con i successivi. In altre parole, facciamo un ciclo su tutti gli elementi tranne l’ultimo. Per ogni elemento, lo leggiamo nella variabile `x`, e poi facciamo il ciclo interno di confronto con i successivi elementi.

Ogni volta che si incontra un elemento minore di `x`, facciamo lo scambio. A questo punto dobbiamo tenere conto che l’elemento i-esimo non è più `x`, dato che è stato sostituito con un altro elemento. Invece di rileggere di nuovo l’elemento i-esimo, mettiamo direttamente in `x` il valore dell’elemento che abbiamo appena scritto nella posizione i del file.

Il programma completo ssort.c è riportato qui sotto.

```
/*
  Selection sort su file binario.
*/

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

int main(int argn, char *argv[]) {
  FILE *fd;
  int size, dim;
  int i, j;
  int x, y;

                /* controllo argomenti */
  if(argn-1!=1) {
    printf("Errato numero di argomenti\n");
    exit(1);
  }

                /* apre il file */
  fd=fopen(argv[1], "r+");
  if(fd==NULL) {
    perror("Errore in apertura del file");
    exit(1);
  }

                /* determina la dimensione del file */
  fseek(fd, 0, SEEK_END);
  size=ftell(fd);
  dim=size/sizeof(int);
  printf("Numero di interi su file: %u/%d = %u\n", size, sizeof(int), dim);


                /* selection sort */
  for(i=0; i<=dim-1-1; i++) {
    fseek(fd, i*sizeof(int), SEEK_SET);
    fread(&x, sizeof(int), 1, fd);

    for(j=i+1; j<=dim-1; j++) {
      fseek(fd, j*sizeof(int), SEEK_SET);
      fread(&y, sizeof(int), 1, fd);

      if(y<x) {
        fseek(fd, i*sizeof(int), SEEK_SET);
        fwrite(&y, sizeof(int), 1, fd);

        fseek(fd, j*sizeof(int), SEEK_SET);
        fwrite(&x, sizeof(int), 1, fd);

        x=y;
      }
    }
  }

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

  return 0;
}
```