File binari, Operazioni sui File binari, Allocazione dinamica della memoria

Vitoantonio Bevilacqua

bevilacqua@poliba.it

Parole chiave: File binari, Funzioni principali file binari, Funzione malloc, Allocazione dinamica vettore, Allocazione dinamica matrice.

1 Principali differenze tra file di testo e file binari

Come visto precedentemente, i file di testo, sono di facile interpretazione per l'utente poiché danno una visione chiara del contenuto: l'utente riesce a rendersi conto del contenuto che è stato usato. Stessa cosa non si può dire per i file binari, che non offrono la stessa chiarezza in quanto scritti in maniera binaria e non formattata (non compare il contenuto in maniera testuale).

I file binari rappresentano la tipologia di file più usati e più utili agli scopi della programmazione per la risoluzione di problemi reali, la lettura di questi avviene bit per bit e non in maniera formattata come per i file di testo, ed ogni singolo bit rappresenta un'informazione interessante allo scopo di poter lavorare sul file stesso.

Esempi di file binari sono: immagini, filmati, suoni, ed altre strutture dati.

1.1 Funzioni per la gestione di file binari

Di seguito sono riportate alcune importanti funzioni per la gestione dei file binari.

La funzione che ci permetterà, in seguito, di fare qualsiasi altra operazione sui file binari è quella che si occupa dell'apertura del file stesso (analoga per i file di testo), la funzione "fopen" che ha le seguente sintassi:

```
FILE *fp;

fp =fopen("percorso file/nomefile.estensione", "modalità apertura");
```

questa funzione permette di associare al file da aprire un flusso specifico.

MODALITA' APERTURA	SIGNIFICATO
“rb”	read (lettura da file)
“wb”	write (scrittura su file)
“ab”	append (scrittura a fine file)
“rb+”	lettura e scrittura
“wb+”	lettura e scrittura
“ab+”	lettura e scrittura a fine file

Esempio:

```
FILE *fp;
fp = fopen(“prova.dat”, “rb”);
```

Una volta aperto il file nella modalità più utile su di esso si potranno compiere diverse azioni, associate a diverse funzioni:

- ftell(fp);

la ftell restituisce un numero che rappresenta il numero del byte su cui (o a partire dal quale) verrà effettuata la prossima operazione;

- fseek(fp,n,macro);

la fseek permette di spostarci all'interno del file binario fp, di un numero n di byte a partire da un numero di byte rappresentato dalla macro;
(SEEK_CUR per il riferimento dalla posizione corrente, SEEK_SET per il riferimento dall’inizio file, SEEK_END per il riferimento dalla fine file)

– - rewind(fp);

la funzione rewind permette di riposizionare l'indicatore di lettura all'inizio del file (al primo byte);

- fwrite(&variabile,sizeof(specificatore_di_tipo),n,fp);

- fread(&variabile,sizeof(specificatore_di_tipo),n,fp);

Le funzioni fwrite e fread sono le funzioni utilizzate rispettivamente per scrivere e leggere in un file binario, queste funzionano in maniera diversa dalle funzioni fprintf e fscanf utilizzate per i file di testo: fwrite scrive sul file fp l'informazione contenuta nella variabile per un numero di byte pari a sizeof()*n, mentre la fread legge dal file fp un numero di byte pari a sizeof()*n e posiziona il contenuto nella variabile;

- feof(fp);

la feof (find end of file) rappresenta l'analoga macro EOF usata per i file di testo. Quindi l'operazione da iterare nel ciclo finche non si ottiene la fine del file richiede la sintassi "while(!feof(fp));"

- fclose(fp);

la funzione fclose interrompe il flusso venutosi a creare nel momento dell'apertura del file;

2 Introduzione al concetto di allocazione dinamica

L'operazione di allocazione dinamica della memoria trova la sua applicazione nel momento in cui è impossibile conoscere a compile-time lo spazio di memoria utile ai fini dell'esecuzione del programma.

Nel momento in cui verrà eseguito il programma sarà deciso (dall'utente o da qualche altro processo) la quantità di memoria da allocare. Questo processo serve ad evitare di sovrastimare, nel momento della compilazione, la quantità di memoria necessaria, ed evitare, quindi, di causare problemi nel momento in cui serva più spazio di quello previsto.

2.1 La funzione "malloc" e la funzione "free"

Per l'utilizzo del processo di allocazione dinamica della memoria occorre fare una premessa: esistono due diversi spazi in memoria per l'esecuzione di un'applicazione

-Memoria "STACK": è la memoria che utilizza la funzione main per le sue variabili e i vari processi da eseguire;

-Memoria "HEAP": è la memoria che viene utilizzata nel momento in cui il programma necessita di altro spazio in memoria, non previsto in fase di compilazione.

La funzione malloc (memory allocation) trova in memoria heap la quantità di spazio richiesto e restituisce l'indirizzo di memoria del primo byte facente parte del blocco di memoria richiesto e trovato (o restituisce NULL se l'operazione non è andata a buon fine).

Esempio:

```
int *p;          //Dichiaro una variabile di tipo puntatore che conterrà
                 //l'indirizzo di memoria restituito da malloc

p = malloc(sizeof(int)); //assegno a p, tramite la funzione malloc,
                         //l'indirizzo di memoria a partire dal quale
                         //potranno essere eseguite le operazioni volute
```

Utilizzando la funzione malloc, si viene a creare, quindi, un collegamento tra la memoria stack e la memoria heap, e si potrà accedere alla memoria allocata dinamicamente tramite l'indirizzo restituito dalla funzione stessa.

N.B. La memoria stack, dopo l'allocazione dinamica, non avrà subìto alcun cambiamento, in quanto tutto ciò che viene allocato dinamicamente si trova nella memoria heap, e nel momento in cui si compiono operazioni su variabili "dinamiche" si va a modificare solo la memoria heap!

Esempio:

```
int *p;
p = malloc(sizeof(int));
*p = 5;   //sto modificando il valore contenuto  all'indirizzo p e non p!!
          //quindi in memoria stack tutto resta uguale
```

Altra funzione associata all'allocazione dinamica è la funzione free che serve a sciogliere il legame venutosi a creare, attraverso la malloc, tra la memoria stack e la memoria heap.

Esempio:

```
free(p);
```

2.2 Allocazione dinamica di un vettore

Spesso nella programmazione si ha bisogno dell'utilizzo di uno o più vettori la cui dimensione non si può sapere a compile-time, ma solo a run-time. Da questo nasce la necessità di dover allocare dinamicamente un vettore.

L'allocazione dinamica di un vettore viene fatta come segue:

```
int *V; //dichiaro una variabile puntatore ad intero che conterrà
        //l'indirizzo della memoria heap a partire dal quale verrà
        //riservato lo spazio per il vettore

V=malloc(sizeof(int)*dim); //uso la funzione malloc per trovare in memoria
                           //heap lo spazio necessario richiesto e assegno
                           //a V l'indirizzo del primo byte
```

A questo punto potrò usare normalmente la sintassi V[i] per accedere ad un elemento del vettore appena creato, dove i rappresenta lo spiazzamento (displacement) dall'indirizzo contenuto in V:

```
V[i]=3;  //vado all'indirizzo V, mi sposto di i*(sizeof(int)) byte e setto
         //il contenuto dell'indirizzo  trovato a 3
```

2.3 Allocazione dinamica di una matrice

Per le stesse ragioni esplicitate per i vettori, anche una matrice ha bisogno di essere allocata dinamicamente. L'allocazione dinamica di una matrice avviene attraverso più passaggi: possiamo pensare una matrice come un vettore colonna i cui elementi siano puntatori ad altri vettori del tipo da noi specificato, quindi:

```
int i;
int **M; //il doppio * rappresenta proprio il fatto che la nostra variabile
          //è un puntatore che punta ad un puntatore di interi

M=malloc(sizeof(int)*colonne); //alloco il vettore colonna fatto di elementi
                                //di tipo puntatore ad interi

for(i=0;i<colonne;i++)
      M[i]=malloc(sizeof(int)*righe);
                  //per ogni elemento di tipo puntatore allocato prima
                  //alloco un altro vettore questa volta di interi
```

Successivamente potrò accedere al generico elemento M[i][j] della matrice nel modo normalmente utilizzato, dove i e j rappresentano sempre il displacement dal primo byte (come visto per i vettori).

Ringraziamenti. Il presente capitolo è stato scritto anche grazie al prezioso contributo degli studenti Pasquale Bonasia, Flavio Palmieri.

Riferimenti

1. Bevilacqua, V.: Dispense Linguaggio C In: http://www.vitoantoniobevilacqua.it
2. http://it.wikipedia.org/wiki/File_binario
3. http://it.wikipedia.org/wiki/Allocazione_dinamica_della_memoria

Appendice: Codice in linguaggio C

```
 //ALGORITMO ALLOCAZIONE DINAMICA MATRICE DA FILE
#include <stdio.h>
#include <stdlib.h>

void allocaMatrice(int ***, FILE *, int);
void leggiFile(FILE *, int*);

void main ()
{
    int **M;
    int ordine;
    int i,j;
    FILE *fp;

    fp=fopen("matrice.txt", "r");

    leggiFile(fp, &ordine);
    allocaMatrice(&M, fp, ordine);
    fclose(fp);
    for(i=0;i<ordine;i++)
    {
        for(j=0;j<ordine;j++)
        {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
    for(i=0; i<ordine;i++)
        free(M[i]);

    free(M);
}


void allocaMatrice(int ***A, FILE *B, int C)
{
    int i,j, elemento;
    *A = malloc(C*sizeof(int*));
```

```
    for(i=0;i<C;i++)
        (*A)[i]=malloc(C*sizeof(int));

    for(i=0;i<C;i++)
    {
        for(j=0;j<C;j++)
        {
            fscanf(B,"%d",&elemento);
            (*A)[i][j]=elemento;
        }
    }
}

void leggiFile(FILE *A, int*B)
{
    fscanf(A, "%d", B);
}
```