La gestione di file in C++

I file sono strutture dati che consentono di conservare permanentemente, su memoria di massa, i nostri dati.
Imparare ad usare i file è quindi INDISPENSABILE per poter programmare.
Per lavorare con i file su disco è necessario includere l'header **<fstream>** che consente di definirne esistenza ed accesso.
I file in C++ possono essere di **testo o binary**.
I file di testo contengono caratteri in formato ASCII, direttamente visibili sullo schermo, e possono essere aperti e modificati da qualsiasi programma editor.
I file binary possono memorizzare qualsiasi tipo di dato. La memorizzazione avviene nello stesso formato in cui sono rappresentati in memoria, senza alcuna conversione. I file binary consentono migliori prestazioni dei file di testo, ma sono meno portabili.

Cosa si intende per ASSEGNAZIONE DEL FILE FISICO

Nessuna "oscura" formula di calcolo degli spazi liberi in memoria: il Sistema Operativo gestisce per noi sia la RAM che la memoria di massa. L'unica cosa che dobbiamo comunicargli è come intendiamo chiamare il nostro fileù, cioè il **nome del file fisico**.
Il nome del file fisico puo' comprendere anche la path, con unità di riferimento, directory e sottodirectory.
Questo ci fa capire perchè quando utilizziamo i file difficilmente abbiamo procedure con parametri: per cambiare l'indirizzo fisico si assegna una nuova path.
Dopo aver individuato il file è necessario, per poter lavorare, aprire la connessione con lo spazio fisico individuato. Questa operazione viene definita **apertura di un file.**

DICHIARAZIONE E APERTURA

Per poter operare su un file è necessario dichiarare uno **stream** ed **associarlo ad un file su disco**, tramite l'**operazione di dichiarazione ed apertura del file**. Con tale operazione si stabilisce la direzione del flusso dei dati: se in lettura, scrittura o entrambe.
Come prima operazione si sceglie il nome da assegnare al file logico, cioè l'identificatore con cui ci riferiamo al file nel programma e poi si procede dichiarando

- **fstream nomefilelogico** per eseguire operazioni I/O
- **ofstream nomefilelogico** per eseguire operazioni di solo Output
- **ifstream nomefilelogico** per eseguire operazioni di solo Input

L'apertura puo' avvenire in diverse modalità con l'istruzione:

nomefilelogico.open(“nomefilefisico”, ios::modalità)

Dove per modalità si intende:

- **ios::out** per scrittura distruttiva: il file associato viene creato, se esiste già il contenuto viene cancellato, la scrittura avviene a partire dall'inizio del file
- **ios::app** per scrittura in coda: il file associato viene creato, se esiste già il contenuto precedente è preservato, la scrittura avviene in append, i nuovi record sono cioè aggiunti alla fine del file.
- **ios::in** per lettura : il file associato deve esistere, la lettura comincia all'inizio del file
- **ios:: binary** apre il file come binario, di default viene considerato text

le varie aperture possono essere combinate con l’ operatore | (or)

TEXT FILE in C++

I file di testo contengono caratteri in formato ASCII, direttamente visibili sullo schermo, e possono essere aperti e modificati da qualsiasi programma editor.

ACCESSO A FILE DI TESTO

Per operare sui file di testo si applicano gli stessi operatori utilizzati per gli stream I/0 standard cin e cout

LETTURA

ifstream f;
f.open (“esempio”, ios::in);
f.get(nomechar);//legge dal file associato allo stream f un solo carattere
f>>stringa; //legge dal file associato allo stream f i caratteri che costituiscono stringa
f>>s1>>s2>>s3;//legge dal file i caratteri che costituiscono ogni dato, gli eventuali spazi tra i dati sono ignorati
getline(f,stringa);//legge l'intera riga del file, compresi gli spazi

per ovviare ad eventuali errori dovuti a problemi nella lettura è possibile utilizzare le funzioni
f.clear;// azzera gli errori
f.ignore(80,'\n');//ignora il resto della riga

SCRITTURA

ofstream f;
f.open (“esempio”, ios::app);
f.put(nomechar)//scrive sul file associato ad f un singolo carattere
f<<stringa;//scrive nel file associato allo stream f i caratteri della variabile stringa
f<<s1<<'\t'<<s2<<'\t'<<s3<<endl;//scrive nel file associato allo stream f le stringhe separando ogni dato con la tabulazione, l'endl invia allo stream line feed+ carriage return e svuota il buffer

Ecco un esercizio di esempio

```
#include<iostream>
#include<fstream>
using namespace std;
```

```
void inserisci();
void stampa();
int main()
{
   int risp;
   do
   {
    cout<<"digita 1 per caricare \n digita 2 per stampare \n digita 3 per uscire \n";
    cin>>risp;
    switch(risp)
    {
     case 1:inserisci();break;
     case 2: stampa(); break;
     }
    }
   while (risp!=3);
}
void inserisci()
{    //dichiarazione in scrittura:  output stream
    ofstream f;
    //apertura per scrittura in coda
    f.open("testo",ios::app);
  string s;
    //lettura di una stringa senza spazi : cin non ammette spazi in lettura
    cout<<"inserisci la stringa da aggiungere al testo"<<endl;
    cin>>s;
    //scrittura nel file notare operatore << di output
    f<<s;
    //chiusura del file
    f.close();
}
void stampa()
    {
    //dichiarazione in lettura :input stream
    ifstream f;
   string s;
    //apertura in lettura
    f.open ("testo",ios::in);
    //ciclo di acquisizione
    while (!f.eof())
    {
        f>>s;
        cout<<s<<endl;

        }
        }
```

Come si puo' notare in esecuzione questo programma genera una duplicazione della stampa dell'ultimo elemento. Modifichiamo la procedura di stampa

```
void stampa()
    {
    //dichiarazione in lettura :input stream
    ifstream f;
    string s;
    //apertura in lettura
    f.open ("testo",ios::in);
    //ciclo di acquisizione
   //sostituisco  while (!f.eof()) con

       while(f>>s)
         cout<<s<<endl;

        }
```

ACCESSO A FILE BINARI

I **file binary** sono file ad accesso diretto utilizzati in modo strutturato (tipizzato) cioè, utilizzati imponendo una **struttura rigida interna**. Tale struttura è definita in base al tipo di dati che il file deve contenere: tutti gli elementi devono essere omogenei e avere lo stesso tipo e **il rispetto della struttura è a carico del programmatore.**
Un file binario deve essere aperto con la modalità **ios::binary**.
Le informazioni gestite con i file binary **non vengono formattate e vengono trattate come blocchi di byte per i quali è necessario specificare indirizzo e dimensione**. Il formato in cui i dati vengono memorizzati nel file strutturato è identico al formato di memorizzazione dei dati in RAM, quindi i trasferimenti tra le due memorie avvengono in modo molto veloce e senza alcuna conversione.
Sebbene i file possano contenere qualsiasi tipo di dato in genere i file strutturati sono organizzati in record (**struct**). **Ogni singolo elemento del file rappresenta un record** e la sua lunghezza dipende dai byte necessari a memorizzare tutti i campi che lo compongono.

Quando dichiariamo un file dobbiamo sempre ricordarci di associare la dichiarazione di un record della stessa struttura. Questo perchè ogni operazione di trasferimento dei dati tra le due memorie (RAM e Massa) avviene sempre un record alla volta, caricando o scaricando il contenuto nella variabile di appoggio.

Usando come esempio la gestione di una agenda agenda vediamo la sintassi della dichiarazione del record:

```
typedef struct contatto
    {
        char nome[30];
        char tel[15];
    };
```

contatto buffer;

- contatto è la struttura del singolo record;
- buffer è la variabile record di appoggio per tutte le operazioni di trasferimento dei dati;

E' importante notare come non vengano usate le stringhe, bensì gli **array di caratteri.** Questo perchè è indispensabile **mantenere fissa in byte la dimensione del record su mass**a.

Per lavorare con i file strutturati dobbiamo conoscere poche, ma fondamentali, istruzioni per scrivere tutti sottoprogrammi di gestione ordinaria della struttura.

lettura e scrittura

Abbiamo visto che prima di leggere o scrivere un file dobbiamo aprirlo. **Dopo l'istruzione di apertura** il puntatore ,cioè il contatore che determina **la posizione del record corrente**, si trova all'**inizio del file** (byte 0). Ogni operazione di I/O determinerà l'**avanzamento (automatico)** del puntatore per un numero di byte pari alla dimensione della variabile letta da file o scritta nel file.

L'accesso ad un file strutturato puo' avvenire sia in modalità sequenziale che in accesso diretto.

Nei file tipizzati in struct le operazioni di trasferimento dei dati tra RAM e massa vanno effettuate su un solo record alla volta.

Per leggere un record (istruzione di input, da massa a RAM) l'istruzione è

- **nomefile.read**((char*)&buffer,sizeof(buffer))

legge dal file associato allo stream nomefile un numero di byte pari alla dimensione di buffer (sizeof buffer) e li memorizza nella variabile di nome buffer ((char *)&buffer).Viene così "copiato" nella variabile chiamata buffer di RAM il record corrente del file per un numero di byte pari alla dimensione di buffer.Ogni lettura va fatta per l'intero record, mai per singolo campo. Naturalmente, una volta che il nostro record si trova ormai in RAM nella variabile buffer possiamo trattare lo struct come in qualsiasi altro programma , operando sui singoli campi

Per scrivere un record (istruzione di output, da RAM a massa) usiamo l'istruzione

- **nomefile.write**((char*)&buffer,sizeof(buffer))

scrive nel file associato allo strem nomefile un numero di byte pari alla dimensione di buffer (sizeof buffer) prelevati dalla variabile buffer ((char *)&buffer), salvando così il dato contenuto dell'intera variabile buffer di RAM nel file di massa assegnato a nomefile, nella posizione individuata dal puntatore di nomefile.
Anche la scrittura viene fatta per l'intero record, mai per singolo campo.

Il cast a char * dell'indirizzo della variabile da trattare è indispensabile.

Ricordiamo che il formato in cui i dati vengono memorizzati nel file strutturato è identico al formato di memorizzazione dei dati in RAM, quindi i trasferimenti tra le due memorie avvengono in modo molto veloce e senza alcuna conversione.

Il formato di memorizzazione spiega perchè con i tali file **non** si possano utilizzare le istruzioni tipiche per gli stream I/O >> e << : i dati memorizzati non sono caratteri ASCII o stringhe come nei text file!

La lettura e la scrittura da un file sono tra le operazioni piu' lente per un computer. Per velocizzarle ad ogni operazione di I/O fatta su massa vengono prelevati piu' record e temporaneamente appoggiati in parti di memoria detta **buffer fisico**. In un buffer fisico sono contenuti piu' record logici(quel record che nel nostro esempio agenda abbiamo chiamato buffer è dunque un buffer logico). Il numero di record logici che un buffer fisico puo' contenere viene chiamato **fattore di bloccaggio**.
Quindi ad ogni lettura o scrittura eseguita in un programma corrisponde un numero ben inferiore di reali accessi fisici, ma l'utente non ne ha alcuna percezione. Questa gestione **non** è compito del programmatore, ma è a carico del S.O. e trasparente per il programmatore. Questo spiega l'importanza dell'istruzione **close**: con la chiusura siamo infatti sicuri che tutti i nostri record siano stati effettivamente trasferiti in massa.

Chiusura

Nomefilelogico.**close**();

Per poter procedere con un esercizio di esempio dobbiamo analizzare alcune funzioni messe a disposizioni dal linguaggio

FUNZIONI UTILIZZABILI

●**eof** restituisce false se il file non è finito
nomefilelogico.eof()

●**seekg** accesso diretto in lettura
●**seekp** accesso diretto in scrittura

nomefilelogico.**seekg(**offset,ios::origine)
nomefilelogico.**seekp**(offset, ios::origine)

dove origine puo' essere ios::cur dal corrente, ios::beg dall'inizio, ios::end dalla fine
in assenza dell' origine spostamento avviene dall'inizio del file e viene detto assoluto

●**tellg** posizione corrente in lettura
●**tellp** posizione corrente in scrittura

n=nomefilelogico**.tellg**()
n=nomefilelogico.**tellp**()

Ora che conosciamo tutte le istruzioni possiamo cominciare a costruire le procedure di gestione ordinaria dei file, quelle cioè necessarie per la gestione di qualsiasi file.

INSERIMENTO DI UN RECORD IN CODA AL FILE

CODICE	NLS
void inserisci()	INSERISCI
{	inizio
ofstream f;	dichiara f in scrittura
f.open("Myagenda.dat", ios::out|ios::app|**ios::binary**);	apri f per scrittura in coda
cout<<"Nome: ";	leggi buffer.nome
cin>>buffer.nome;	
cout<<"Telefono: ";	leggi buffer.tel
cin>>buffer.tel;	
f.write((char*)&buffer, sizeof(buffer)); /*scrivi nel file f il record che hai dichiarato di nome buffer */	scrivi in f buffer
f.close();	chiudi f
}	fine

STAMPA DI TUTTO IL FILE

Per stampare tutto il file dobbiamo acquisire ogni singolo record e visualizzarne i campi

CODICE	NLS
void stampa()	STAMPA
{	inizio
ifstream f;	dichiara f in lettura
f.open("Myagenda.dat", ios::in|**ios::binary**);	apri f in lettura
if (!f) cout<<"Error";	se f non si apre stampa "Error"
else	altrimenti
while(**f.read**((char*)&buffer,sizeof(buffer))) cout<<"Nome:"<<buffer.nome<<"Telefono:" <<buffer.tel<<endl; /* fino a che la lettura riesce leggi dal file il record che hai chiamato buffer*/	mentre (leggi da f buffer) stampa buffer.nome, buffer.tel
f.close();	chiudi f
}	fine

RICERCA UNO : CERCA UN TELEFONO DATO IN INPUT IL NOME

/**
commento sul confronto tra stringhe come array di caratteri
Si usano array di caratteri quindi il confronto non si puo fare con == ma con strcmp.
se strcmp(stringa1,stringa2) è = a 0 le due stringhe sono uguali
se strcmp(stringa1,stringa2) >= 0 è maggiore stringa1
se strcmp(stringa1,stringa2) <0 è maggiore stringa 2
**/

CODICE	NLS
void ricerca_tel()	RICERCA TELEFONO
{	inizio
char k[30]; //variabile k stesso tipo di buffer.nome	dichiara k
cout<<"Nome contatto: ";	leggi k
cin>>k; //inserisci il nome da ricercare	
ifstream f	dichiara file f in lettura
f.open("Myagenda.dat", **ios::in**|**ios::binary**);	apri f in lettura
while(**!f.eof() && strcmp(buffer.nome,k)!=0**) f.read((char*)& buffer, sizeof(buffer));	mentre (non è finito f e buffer.nome<>k) leggi da f buffer
if**(strcmp(buffer.nome,k)==0)** cout<<"Il numero di "<<k<<" e': "<<buffer.tel;	se(buffer.nome=k) stampa buffer.tel
else cout<<"Contatto non presente"<<endl;	altrimenti stampa "Contatto non presente"
f.close();	chiudi f
}	fine

MODIFICA IL TELEFONO DATO IN INPUT IL NOME

Dopo aver trovato il record da modificare prima di scrivere nel file è necessario riposizionare il file tornando indietro di una posizione (cioè un record)

CODICE	NLS
void **modifica_telefono()**	**MODIFICA_TELEFONO**
{	inizio
char k[30]; //variabile k stesso tipo di buffer.nome	dichiara k
cout<<"Nome contatto: "; cin>>k; //inserisci nome di cui modificare il telefono	leggi k

fstream f	dichiara file f in lettura e scrittura
f.open("Myagenda.dat", **ios::in|ios::out**|ios::binary);	apri f in lettura e scrittura
while(!f.eof() && strcmp(buffer.nome,k)!=0) f.read((char*)& buffer, sizeof(buffer));	mentre (non è finito f e buffer.nome<>k) leggi da f buffer

if(strcmp(buffer.nome,k)==0)
{
//calcola la nuova posizione
long int **pos=f.tellg()**;
pos=pos-sizeof(buffer);
//torna indietro di una posizione
f.seekp(pos,ios::beg);
cout<<"inserisici nuovo telefono";
cin>>buffer.tel;
f.write((char*)&buffer,sizeof(buffer));
}
	se(buffer.nome=k)
	inizio
	
	pos=posizione corrente di f
	pos=pos-dimensione di un record
	
	vai in f alla posizione pos
	
	leggi buffer.tel
	scrivi in f il buffer
	fine se

f.close();	chiudi f
}	fine

ELIMINA UN RECORD DAL FILE (con file di appoggio)

La funzione è organizzata in due blocchi, nel primo leggiamo dal file f e scriviamo nel file di appoggio tutti i record diversi da quello da cancellare, nel secondo dopo aver ripulito il file f aprendolo in out, riscriviamo tutti i record salvati in appoggio

CODICE	NLS
void cancella()	CANCELLA
{	inizio
{	inizio prima parte
ofstream appo;	dichiara file appo in scrittura
ifstream f;	dichiara file f in lettura
char k[30];	dichiara k
cout<<"Nome contatto da cancellare: "; cin>>k;	leggi k
appo.open("appo.dat",**ios::out**|ios::binary);	apri appo in scrittura distruttiva
f.open("Myagenda.dat", **ios::in**|ios::binary);	apri f in lettura

while(**f.read**((char*)&buffer, sizeof(buffer)))
{
if(strcmp(buffer.nome,k)!=0)
appo.write((char*)&buffer,sizeof(buffer))
}
	mentre (leggi da f buffer)
	inizio
	se (buffer.nome=k)
	scrivi in f buffer
	fine mentre

f.close();	chiudi f
appo.close();	chiudi appo
}	fine prima parte
{	inizio seconda parte
ifstream appo;	dichiara appo in lettura
ofstream f;	dichiara f in scrittura
appo.open("appo.dat",**ios::in**|**ios::binary**);	apri appo in lettura
f.open("Myagenda.dat", **ios::out**|**ios::binary**);	apri f in scrittura distruttiva
while(**appo.read**((char*)&buffer, sizeof(buffer)))	mentre (leggi da appo buffer)
f.write((char*)&buffer,sizeof(buffer));	scrivi in f buffer
f.close();	chiudi f
appo.close();	chiudi appo
}	fine seconda parte
}	fine