I File in C++

Introduzione

- Per poter mantenere disponibili i dati tra le diverse esecuzioni di un pgm (persistenza dei dati) è necessario poterli archiviare su memoria di massa (nastri, dischi magnetici, ...).

I file

- Un **file** è in insieme di dati correlati, identificato da un nome, memorizzato permanentemente su un supporto di memoria non volatile e avente vita indipendente dal pgm (o dai pgm) utilizzato/i per la sua creazione e/o modifica.

File system

- Il **file system** è la componente del sistema operativo che gestisce i dati presenti nella memoria non volatile del computer.
- Esistono diversi tipi di file system, ma tutti organizzano i contenuti in cartelle (directory) e file.

Organizzazione e modalità di accesso

- I dati in un file sono sempre memorizzati come una sequenza di byte.
- Organizzazione sequenziale: organizzazione che permette di accedere ai dati nello stesso ordine in cui questi sono stati scritti. Per leggere un dato è necessario leggere prima tutti quelli che lo precedono, senza possibilità di tornare indietro (accesso sequenziale).
- Organizzazione relative: organizzazione che permette di accedere ai dati specificando la loro posizione all'interno del file (accesso diretto).

File di testo

I file di testo

- sono file di caratteri, organizzati in linee
- ogni linea è terminata da una marca di fine linea (‘\n’).

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).

File di testo

- Si noti che lo standard input (in genere la tastiera) e lo standard output (normalmente la finestra di testo sul video) sono visti dal programma come file sequenziali di testo.

I canali per l'I/O in un programma

- Un programma comunica con l'esterno attraverso strutture logiche dette stream o flussi.
- Un programma, in genere, preleva i dati di ingresso dallo standard input ed emette i dati di uscita sullo standard output. Questi "canali" possono essere visti come flussi di caratteri sui quali è possibile leggere o scrivere, rispettivamente.
- I flussi vengono associati:
  - o a dispositivi fisici (video, tastiera, stampanti),
  - o a file residenti in memoria secondaria.
- Si può accedere ai dati di un flusso in maniera:
  - sequenziale: si opera sempre sul dato successivo al dato su cui si è operato precedentemente;
  - diretta: si può operare su un dato qualunque del flusso.

Input/Output su dispositivi fisici

- Per le operazioni di ingresso/uscita da/verso dispositivi fisici si fa uso di due flussi predefiniti:
  - l'oggetto cin della classe istream che è associato al dispositivo standard di ingresso (tastiera);
  - l'oggetto cout della classe ostream che è associato al dispositivo standard di uscita (video);
- Per questi flussi ha senso solamente l'accesso sequenziale.
- Gli operatori di base sono >> (ingresso) e << (uscita) che sono definiti per i tipi fondamentali (int, char, float e double) e per le stringhe.
- Nota: L'operatore >> scarta tutti gli spazi, le tabulazioni e il carattere di fine riga inseriti in ingresso.

Lettura di dati da tastiera

- Per leggere da tastiera un intero `i`, un numero reale `z` oppure una stringa `str`, il C offre la funzione `scanf` (di input formattato):

```
scanf ("%d %f %s", &i, &z, str);
```

- Il C++ permette simili operazioni in modo più semplice, interpretando automaticamente il tipo del dato a partire da quello della variabile in cui il dato viene scritto:

```
cin >> i >> z >> str;
```

C: #include<stdio.h>

C++: #include <iostream>

Stampa sul monitor

- Per stampare sul monitor (standard output) un intero `i`, un numero reale `z` oppure una stringa `str`, in C si usa la funzione `printf` (di output formattato):

```
printf ("%d %f %s", i, z, str);
```

- In C++ si utilizza la funzione `cout`, che interpreta automaticamente il tipo del dato a partire da quello della variabile da stampare:

```
cout << i << z << str;
```

Lettura di caratteri

- La lettura di un carattere singolo, spazi inclusi, si effettua in C con la funzione `getchar()`:

```
char ch;
ch = getchar();
```

Il corrispondente in C++ è la funzione `cin.get()`:

```
ch = cin.get();
```

Scrittura di caratteri

- C
  - char car;
  - putchar(car);
- C++
  - cout.put(car);

Esempio

- Prova su computer il seguente ciclo:

```
while((car = cin.get()) != EOF)
    cout.put(car);
```

NOTA: Il valore della costante predefinita EOF è -1.

Scrittura con formattazione

- La stampa sul monitor di un intero `i` in ottale o in esadecimale avviene in C con opportuni caratteri di conversione applicati alla funzione `printf()`:

```
printf ("%o %h", i, i);
```

- La stessa formattazione si ottiene in C++ con la sequenza:

```
cout << oct << i << hex << i;
```

Nota: codice di formato esadecimale è %x e non %h

Scrittura di numeri reali

- In C la stampa di un numero reale `z` può essere effettuata specificando negli argomenti di `printf()` il numero di caratteri totali e il numero di cifre decimali desiderati:

```
printf ("%10.4f", z); /* stampa z su 10
caratteri con 4 cifre decimali */
```

- In C++ simili formattazioni, con risultato generalmente diverso, si ottengono ad esempio con:

```
cout << cout.precision(4) << z;
```

Controllo sui formati

- Il formato di uscita dipende da una serie di parametri che hanno dei valori di default ma possono essere cambiati mediante manipolatori di formato.

Manipolatori

- Potrebbero richiedere di includere `<iomanip>`

Manipolatore	Significato
`flush`	Svuota il buffer
`endl`	Aggiunge un carattere di fine linea e svuota il buffer
`hex`	Gli interi vengono formattati in base 16
`oct`	Gli interi vengono formattati in base 8
`dec`	Gli interi vengono formattati in base 10

Altri manipolatori

Manipolatori

- Manipolatori di campo

Manipolatore	Significato
`setw(w)`	Setta l'ampiezza del campo (default = 0)
`setfill(c)`	Setta il carattere di riempimento (default = *spazio*)
`left`	Il testo viene allineato a sinistra nel campo
`right`	Il testo viene allineato a destra nel campo (default)

Estensione dell'I/O ai tipi definiti dall'utente

- E' comune in C++ estendere le operazioni di I/O anche ai tipi definiti dall'utente (con il costrutto di classe).

  Es. frazione f;

  cin>>f;

  cout<<f;

- Per rendere lecita una tale sintassi è necessaria la ridefinizione degli operatori << e >> sulle classi.

Operazioni su file

- Le operazioni che la gestione dei file comunemente prevede sono le seguenti:
  - **Apertura**: il programma comunica al sistema operativo la necessità di accedere ad uno specifico file.
  - **Lettura**: il programma richiede dei dati dal file come input.
  - **Scrittura**: il programma intende scrivere dei dati in un file come output.
  - **Chiusura**: il programma comunica al sistema operativo che non ha più necessità di accedere al file.

Buffer

- E’ l’area di memoria volatile riservata dal S.O. all’atto dell’apertura di un file per gestire lo scambio dei dati tra programma e memoria non volatile; l’operazione di chiusura di un file aperto in scrittura fa sì che eventuali dati contenuti nel buffer siano copiati nella memoria non volatile prima che questo venga rilasciato.

Gestione dei file in C++

- Per le operazioni di ingresso/uscita da/verso file è sufficiente definire un oggetto delle classi ifstream, ofstream o fstream.
- Prima di operare su un file bisogna aprirlo con la funzione open() specificando il file fisico associato e la modalità di apertura:
  - ios::in (apertura in lettura),
  - ios::out (apertura in scrittura),
  - ios::app (apertura di scrittura in append)
  - ios::bynary (apertura in modalità binaria, la modalità testo è quella di default))
  - ios::in | ios::out (apertura in lettura/scrittura)
- Per default, un oggetto di ifstream viene aperto in lettura mentre un oggetto di ofstream viene aperto in scrittura.
- In C per operare sui file devi includere <stdio.h>, in C++ devi includere <iostream> e <fstream>

Controllo sui flussi

- Per fare controlli sui flussi è possibile usare la funzione:
  - int eof(): restituisce il valore di un indicatore di stato che viene posto a 1 quando si incontra la fine del file.

Input/Output su file

Esempio:

```
ifstream infile;
ofstream outfile;
....
   infile.open("uno.txt");
   outfile.open("due.txt",ios::out);
```

- Nota: per operare su un file possiamo applicare tutte le funzioni di I/O viste finora.

Esempi di programmi che operano su file

```
// Programma che scrive i dati letti  da input sul file //dati.txt
#include<fstream>
#include <iostream>

using namespace std;

int main(){
ofstream outfile;                    // denota il file sul quale scrivere i dati
char c;
  outfile.open("dati.txt");
  if (!outfile)  {
    cout << "Errore in apertura del file dati”<< endl;
  } else  {
      while((c = cin.get()) != '\n')
         outfile.put(c);
      outfile.close();
  }
}
```

Esempi di programmi che operano su file

```
// Programma che copia il  contenuto di un file in un altro
int main(){
fstream infile;   // file di ingresso
fstream outfile; // file di uscita
char c;

        infile.open("dati.txt",ios::in);
        if (!infile){
            cout << "Errore in apertura del file dati”<< endl;
            return 1;
        }
        outfile.open("copia.txt",ios::out);
        if (!outfile){
           cout << "Errore in apertura del file copia” << endl;
           return 1;
        }
```

Esempi di programmi che operano su file

```
    infile.get(c);                    //concetto di cursore
    while(!infile.eof())  {
        outfile.put(c);
        infile.get(c);
    }
    outfile.close();
    infile.close();
}
```

O in modo equivalente …

```
…
while(infile.get(c)) {
    outfile.put(c);
}
…
```

Nota: il metodo (funzione) get() della classe fstream restituisce il valore 0 quando si incontra la fine del file.

Apertura e chiusura

<tipo_file> nomeInterno(nomeEsterno [,specificatori]);

oppure

<tipo_file> nomeInterno;

nomeInterno.open(nomeEsterno [,specificatori]);

<tipo_file> è una delle classi ifstream, ofstream o fstream.

per la chiusura di un file si usa in tutti i casi:

nomeInterno.close();

Controllo apertura

- Subito dopo l'apertura è possibile controllare se essa è andata a buon fine nel modo seguente:

```
if (!fileDaLeggere){
   cout << "Errore nell'apertura del file"<<endl;
}
```

Valori di default

- Se gli specificatori (ios::in, …) non vengono utilizzati si usano i seguenti valori di default:
  - l'apertura è in modalità testo
  - in input, se il file non esiste, l'apertura dà errore
  - in output, se il file non esiste viene creato, se esiste viene cancellato e sovrascritto
  - per ifstream è ios::in
  - per ofstream è ios::out
  - per fstream non vi è default, il modo va esplicitato

Esercizio svolto

Si assuma che una matrice sia memorizzata in un file per righe, in accordo al seguente formato:

r c a_{11} a_{12} … a_{1n} a_{21} a_{22} … a_{2n} … a_{m1} a_{m2} … a_{mn}

con aij elemento generico della matrice.

Scrivere un programma che:

- legge una matrice di double da un file di ingresso (che si assume nel formato corretto);
- scrive su standard output il valore della somma degli elementi della matrice;
- scrive in un file di uscita la matrice trasposta.

I nomi dei file di ingresso e di uscita sono letti da standard input e sono costituiti da un numero di caratteri pari ad, al più, MAX_LUN_NOME.

Soluzione

```
const int M = 10;
cont int MAX_LUN_NOME = 30+1;
int main() {
fstream ingr, usc;
char nomeIngr[MAX_LUN_NOME],      // stringhe per
nomeUsc[MAX_LUN_NOME];            // i nomi dei file
double m[M][M], t[M][M];          // la matrice e la trasposta
int r, c;

                                  // si leggono i nomi dei file
      cin >> nomeIngr >> nomeUsc;

                                  // si aprono i file in lettura e scrittura, rispettivamente
      ingr.open(nomeIngr, ios::in);

                                  // Si legge la matrice dal file d'ingresso
                                  // 1) si leggono il numero di righe ed il numero di colonne
      ingr >> r >> c;

                                  // 2) si leggono, per riga, gli elementi della matrice
      for (int i = 0; i < r; i++)
          for (int j = 0; j < c; j++)
              ingr >> m[i][j];
      ingr.close();               // il file d'ingresso non serve più
```

```
                        // Si scrive la somma degli elementi della matrice su std. output
cout << sommaElementi(m, r, c);
                                    // Si calcola la trasposta
trasposta(m,t, r, c);

usc.open(nomeUsc, ios::out);
                                    // Si salva la matrice trasposta sul file di uscita
                                    // 1) Si salvano il numero di riga e di colonna
usc << c <<'\t'<<<< r;
                                    // 2) Si salvano, per riga, gli elementi della matrice
for (int i = 0; i < c; i++)
     for (int j = 0; j < r; j++)
              usc << '\t'<<t[i][j];
usc.close();                        // Il file di uscita non serve più
}
```

```
double sommaElementi(const double m[][M], int r, int c) {
double sommaElem = 0.0;
   for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
             sommaElem += m[i][j];
   return sommaElem;
}
```

```
void trasposta (const double m[][M], double t[][M], int r, int c){
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            t[j][i] = m[i][j];
}
```

File di testo formattati

- Esempio: file che contiene i valori della temperatura min e max giornalieri.

```
12      20
10      18
11      19
...
```

File di testo formattati

```
…
in>>min>>max;
while(!in.eof()) {
    //elaborazione
    in>>min>>max;
}
…
```

```
…
while(in>>min>>max) {
    //elaborazione
}
…
```

Nota: (in>>min>>max) ritorna il valore 0 se il file è finito.

File di testo formattati

```
…
in>>min>>max;
while(!in.eof()) {
    media = (min + max)/2.0;

    out<<media<<endl;

    in>>min>>max;
}
…
```

File di testo formattati

- Esempio visitatori mostra: sesso età

M 24
M 38
F 24
F 32
M 45
…

File di testo formattati

int eta;

char sesso[2]; //stringa e non carattere

in>>sesso>>eta;

out<<sesso<<‘\t’<<eta<<endl;

Lettura/scrittura file (di testo)

- Lettura/scrittura di caratteri
- Lettura/scrittura con formato
- Lettura/scrittura di stringhe di caratteri (da fare)

Osservazione

- I file sequenziali di testo rappresentano il modo più semplice per memorizzare dati sulla memoria non volatile; la loro struttura non è molto sofisticata e le loro prestazioni in termini di funzionalità offerte sono estremamente ridotte.

Osservazione 2

- I file di testo spesso costituiscono la maniera più semplice , e talvolta l'unica, per scambiare dati tra applicazioni s/w che utilizzano formati di file dati non compatibili tra loro.

Osservazione 3

- In C/C++ carattere di “andata a capo” è ‘\n’.
- Su file il ritorno a “capo” è dato dai caratteri ASCII <CR> (codice decimale 13) e <LF> (codice decimale 10).
- Quando scrivi un file di testo ad es. con Blocco Note di Windows (o un qualsiasi altro Editor di testo) o un Pgm applicativo ricordati quando scrivi l’ultima riga di andare poi a capo.

File in formato CSV (Comma Separated Values)

- Sono spesso utilizzati per l'importazione e l'esportazione di una tabella di dati da applicazioni come i gestori di fogli di calcolo e di database.
- Non esiste uno standard formale che definisce questa tipologia di file. Generalmente:
  - Riga tabella ←→ Linea di testo (seguita dal carattere di "andata a capo".
  - Carattere separatore colonne/campi (es. virgola oppure punto e virgola, oppure …)
  - Riga di intestazione

Esempio

Cognome	Nome	Indirizzo	CAP	Città
Rossi	Marco	Via Roma, 12	57100	Livorno
Neri	Andrea	Via del Mare, 15	56100	Pisa
Bianchi	Giovanni	Via Ferruccio, 7	55100	Firenze

Esporta

- La tabella precedente esportata in un file in formato CSV potrebbe assumere il seguente formato:

- Cognome;Nome;CAP;Città
- Rossi;Marco;Via Roma, 12;57100;Livorno
- …

Analisi problema

- Spazi bianchi come separatori
- Es. Indirizzo
  - Via Roma, 12 → 3 spazi bianchi
  - Via Antica Roma, 12 → 4 spazi bianchi
- Leggere ogni singola riga come un'unica stringa di caratteri e individuare successivamente le singole componenti (Cognome, Nome, …) mediante una funzione che esamina la stringa alla ricerca del carattere di separazione, in modo da poterla suddividere nei vari elementi.

Lettura stringhe in C++

- #define MAX_LEN 256+1
- char riga[MAX_LEN];
- ifstream rigaTab;

- rigaTab.getline(riga, MAX_LEN);

Funzione estraiElemento()

void estraiElemento(char rigaIn[], char elemento[]) {

- ricerca carattere separatore o fine stringa (1° parametro)
- copia elemento (2° parametro)
- eliminazione della stringa dell'elemento trovato(cancellazione fisica)

}