Corso sul linguaggio C++

Modulo 6

2 - Struttura file binari

Prerequisiti

- Operazioni base su file
- Memorie secondarie

Argomenti

- File binari
- Operazioni sui file
- Apertura file
  - classe **ifstream**
  - classe **ofstream**
  - classe **fstream**
  - considerazioni
- Lettura da file
- Scrittura su file
- Chiusura del file
- Test di controllo file
- Accesso diretto a record
- Esempi
- File binario di numeri
  - ScriviFile.cpp
  - LeggiFile.cpp
  - Eliminazione.cpp
  - AppendiFile
  - AccessoDiretto
- **get()** e **put()** su file binari
- File binario di **struct**
  - Lettura
  - Scrittura
  - Eliminazione
  - Stampa

Introduzione

In questa Unità vediamo gli strumenti per l'utilizzo dei **file binari** in C++.

I file binari si utilizzano di solito quando abbiamo a che fare con archivi i cui record hanno **lunghezza fissa** ed offrono una più efficiente memorizzazione.

Per trattare i file binari si usano le classi note: **fstream**, **ofstream** e **ifsteram**

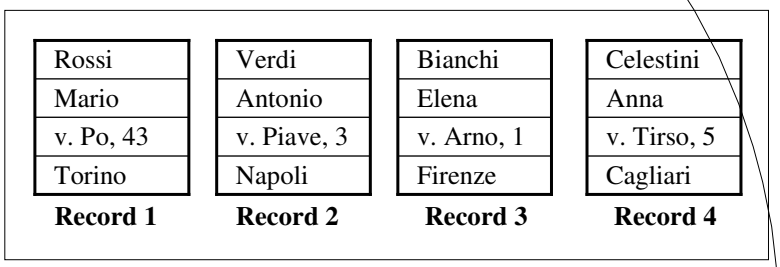
File binari

I **file binari**

- sono costituiti da sequenze di byte, organizzati in record logici
- non sono leggibili, non sempre sono portabili, ma sono veloci nell'I/O

Ad es. archivi di **struct**

File sequenziale



Operazioni sui file

Le principali **operazioni logiche** sui file binari sono le stesse dei file testo:

- Apertura file (in lettura, scrittura o in accodamento)
- Lettura da file
- Scrittura su file
- Chiusura del file
- Test di fine file
- Accesso diretto

Apertura file

L'apertura di un file in C++ può essere fatta nei modi seguenti:

- Apertura file in **lettura**
- Apertura file in **scrittura**
- Apertura file in **accodamento**
- Apertura in **lettura/scrittura**

Apertura file
classe ifstream

L'apertura in **lettura** di uno stream in C++ può essere fatta mediante oggetti di classe **ifstream** nei modi seguenti:

ifstream *nomestream*; *nomestream*.**open** (*nomefile*, **ios::binary**);	Istanzia l'oggetto *nomestream* Apre *nomestream* in lettura **binaria**
ifstream *nomestream*; *nomestream*.**open** (*nomefile*, **ios::in** | **ios::binary**);	Istanzia l'oggetto *nomestream* Apre *nomestream* in lettura **binaria**

Apertura file
classe ofstream

L'apertura in **scrittura** di uno stream in C++ può essere fatta mediante oggetti di classe **ofstream** nei modi seguenti:

ofstream *nomestream*; *nomestream*.**open**(*nomefile*, **ios::out** | **ios::binary**);	Istanzia l'oggetto *nomestream* Apre *nomestream* in scrittura **binaria**
ofstream *nomestream*; *nomestream*.**open**(nomefile, **ios::binary**);	Istanzia l'oggetto *nomestream* Apre *nomestream* in scrittura **binaria**

Apertura file
classe ofstream

L'apertura in **accodamento** di uno stream in C++ può essere fatta su oggetti di classe **ofstream** nei modi seguenti:

ofstream *nomestream*; *nomestream*.**open**(*nomefile*, **ios::app** | **ios::binary**);	Istanzia l'oggetto *nomestream* Apre *nomestream* in accodamento **binario**

Apertura file
classe fstream

L'apertura in **lettura/scrittura** di uno stream in C++ può essere fatta mediante oggetti di classe **fstream** nei modi seguenti:

fstream *nomestream*; *nomestream*.**open**(*nomefile*, **ios::in** | **ios::binary** | **ios::out**);	Istanzia l'oggetto *nomestream* Apre *nomestream* in lettura/scrittura **binaria**

Apertura file
considerazioni

Quindi:

1. Se lo stream deve essere aperto in lettura si deve istanziare la classe **ifstream** o **fstream**
2. Se lo stream deve essere aperto in scrittura o in accodamento si deve istanziare la classe **ofstream** o **fstream**
3. Se lo stream deve essere aperto in lettura/scrittura va istanziato dalla classe **fstream.**
4. Se si usa la classe **fstream**, le modalità di apertura (lettura, scrittura, accodamento, bnario) devono essere esplicitate tramite la costante **ios**.

Lettura da file

L'operazione di lettura da file binario si esegue come indicato:

nomestream.**read**((**char**) **buffer*, **sizeof**(*buffer*)	Legge da *nomestream* **sizeof**() byte e li pone in *buffer*
int n = *nomestream*.**get** ()	Legge da *nomestream* il dato intero *n*

Scrittura su file

L'operazione di scrittura su file binario si esegue come indicato

nomestream.**write**((**char**) **buffer*, **sizeof**(*buffer*)	Scrive su *nomestream* **sizeof**() byte presenti in *buffer*
nomestream.**put** (**int** n)	Scrive su *nomestream* il dato intero *n*

Chiusura del file

L'operazione di chiusura del file binario si esegue come indicato

nomestream,.**close**()	Chiude *nomestream* e rilascia il buffer

Test di controllo file

L'operazione di **test di fine file** si esegue come indicato

if (!*nomestream*) ….	Testa la fine del file *nomestream*

L'operazione di **test dell'esistenza del file** può essere espressa con

if (*nomestream*.**fail**()) ….	Testa l'esistenza del file *nomestream*

Accesso diretto a record

L'operazione di **accesso diretto** si esegue con:

*nomestream.***seekp**(**long** *pos****sizeof**(*rec*), **ios**:*origin*);	Avanza l'indicatore di un numero di byte pari alla posizione richiesta *pos*, per l'occupazione del record *rec.*
*nomestream.***seekg**(**long** *pos****sizeof**(*rec*),);	*origin* indica il punto da cui inizia l'accesso diretto e può valere: • **beg**: inizio stream; • **end**: fine stream • **cur**: posizione corrente Dopo la **seekg() o seekp()** occorre una lettura per portare il record in memoria.

Esempi

Vediamo ora alcuni esempi di codifica di programmi che fanno uso di file binari.

Dapprima vediamo come elaborare numeri su file binari.

Successivamente, si trattano file binari di record e l'accesso diretto.

File binario di numeri

Presentiamo le varie operazioni come piccoli programmi a sé stanti. Il lettore, successivamente, potrà creare un'applicazione che li integri come funzioni legate da un menu operativo.

Esaminiamo i seguenti programmi:

- ScriviFile.cpp
- LeggiFile.cpp
- Eliminazione.cpp
- AppendiFile.cpp
- AccessoDiretto.cpp

File binario di numeri

ScriviFile.cpp

```
Inclusione librerie <cstdlib>, <iostream>, <fstream>
#define filename "archivio.dat"
using namespace std;
typedef int dati;
int main()
{   dati n;
    ofstream fout;
    fout.open (filename, ios::binary);
    cout<<"Inserire valori (Ctrl-Z per finire): "<<endl;
    while (cin>>n)
        fout.write ((char*) &n, sizeof (int));
    fout.close();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Per la scrittura dei dati si usa il metodo **fout.write()** inserito nel ciclo.
La condizione di controllo del ciclo.è data da Ctrl-Z.

Per correttezza, sarebbe opportuno testare l'esistenza del file, per far decidere all'utente se riscrivere o accodare.

File binario di numeri
LeggiFile.cpp

```
Inclusione librerie <cstdlib>, <iostream>, <fstream>
#define filename "archivio.dat"
using namespace std;
typedef int dati;
int main()
{  dati n;
   ifstream fin;
   fin.open(filename, ios::binary);
   while (fin.read((char *) &n, sizeof(int)))
     cout<<n<<endl;
   fin.close();
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Per la lettura dei dati si usa il metodo **fin.read()** inserito nella condizione di controllo del ciclo.

File binario di numeri
Eliminazione.cpp

```
Inclusione librerie <cstdlib>, <iostream>, <fstream>
#define filename "archivio.dat"
using namespace std;
typedef int dati;
int main()
{  dati n, n1;
   long pos;
   fstream fin;
   fstream faux;
   fin.open(filename, ios::in | ios::binary);
   immettere posizione pos da eliminare;
   Se trovata      copia in faux tutti i valori tranne quello in posizione pos
   Altrimenti     stampa messaggio di errore;
   system("PAUSE");
   system("del temp");      // cancellazione file temporaneo
   return EXIT_SUCCESS;
}
```

Per la eliminazione si usa un file di appoggio *faux* in cui si copiano i record ad eccezione di quello da eliminare.

Successivamente *faux* viene copiato in *fin*v sfruttando il comando **copy** del sistema operativo.

File binario di numeri
Eliminazione.cpp

```
copia in faux tutti i valori tranne quello in posizione pos
    cout<<"Posizione da eliminare: ";
    cin>>n1;
    pos=(n1-1)*sizeof(int);
    fin.seekg(pos);              // accesso diretto
    fin.read((char*) &n, sizeof(int));
    if (fin)                     // se il record esiste…
    {     fin.seekg(0);          // …torna all'inizio
          int del=n;
          faux.open("temp", ios::out | ios::binary);
          while(fin.read((char*) &n, sizeof(int)))
              if (n!=del)
                    faux.write((char*) &n, sizeof(int));
          fin.close();
          faux.close();
          system("copy temp archivio.dat>null");
    }
```

La funzione copia i valori da *fin* in *faux* tranne il valore di cui si è richiesta la cancellazione.

File binario di numeri
AppendiFile.cpp

```
Inclusione librerie <cstdlib>, <iostream>, <fstream>
#define filename "archivio.dat"
using namespace std;
typedef int dati;
int main()
{  dati n;
   fstream fout;
   fout.open(filename, ios::binary | ios::app |ios::out);
   cout<<"Inserire valori (Ctrl-Z per finire): "<<endl;
   while (cin>>n)
      fout.write((char*) &n, sizeof(int));
   fout.close();
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

La funzione carica valori nel file e se non esisteva lo crea.

File binario di numeri

AccessoDiretto.cpp

```
incluisione <cstdlib>, <iostream>, <fstream>
#define filename "archivio.dat"
using namespace std;
typedef int dati;
int main()
{  dati n, n1;
   long pos;
   ifstream fin;
   fin.open(filename, ios::binary | ios::in);
   cout<<"Posizione da cercare: ";   cin>>n1;
   pos=(n1-1)*sizeof(int);
   fin.seekg(pos);
   fin.read((char*) &n, sizeof(int));     // lettura necessaria dopo la seek()
   if (fin)       cout<<n<<endl;
   else           cout<<"Non trovato"<<endl;
   fin.close();
   system("PAUSE");
   return EXIT_SUCCESS;
}
```


get() e **put()** su file binari

Vediamo un esempio di uso di **get()** e **put()** in un programma che legge da input numeri interi fino alla pressione di Ctrl-Z, li registra in un file che viene poi stampato

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#define datafile "numeri.dat"
using namespace std;
void stampa (char filename[12]);
int main()
{  fstream file;
   int n;
   cout<<"Questo programma crea un file di interi NUMERI.DAT\n\n\n";
   file.open(datafile, ios::out | ios::binary);
   while (cin>>n)
   {   file.put(n);
       cout<<"Inserisci un numero (Ctrl-Z per terminare): ";
   }
```

get() e **put()** su file binari

```
 file.close();
 stampa(datafile);
 system ("pause");
 return EXIT_SUCCESS;
}
void stampa (char filename[12])
{  int n;
   ifstream fstampa;
   fstampa.open(filename, ios::in);
   while (!fstampa.eof())
   {
      n=fstampa.get();
      if (!fstampa.eof())
      cout<<n<<endl;
   }
   fstampa.close();
}
```

Il ciclo di stampa procede fino a che fstampa.**eof()** diventa **true**.

File binario di **struct**

```
#include <iostream>
#include <fstream>
typedef struct
{
    int ID;
    char cognome[20];
    char interno[10];
} scheda;
```

Negli esempi di file binari che seguono, facciamo uso di una **struct**. In questo caso, poiché una **struct** è di lunghezza fissa, è possibile usare l'accesso diretto.

File binario di **struct**

Lettura

```
Inclusione librerie
using namespace std;
const char filename[10]="Dati";
typedef struct { … } scheda;
int ricerca(int id, scheda &r);
scheda rec;
ifstream filein;
ofstream fileout;
int main()
{   int id;
    long pos;
    filein.open(filename, ios::in|ios::binary);
    if (filein.fail())  cout<<"File inesistente!";
    else      Immetti codice e visualizza esito
    system("Pause");
    return EXIT_SUCCESS;
}
```

Funzione per visualizzare i dati relativi ad un record di cui si immette il codice.

Si apre il file in lettura e se ne controlla l'esistenza.

File binario di **struct**

Lettura

```
Immetti codice e visualizza esito

    cout<<"ID.....:";
    cin>>id;
    filein.close();
    pos = ricerca(id, rec);       // trova posizione del record
    if (pos)                      // trovato  pos !=0
    {       cout<<"Cognome.:"<<rec.cognome<<endl;
            cout<<"Interno.:"<<rec.interno<<endl;
     }
     else  cout<<"Inesistente"<<endl;
     filein.close();
```

File binario di **struct**

Lettura

```
int ricerca(int id, scheda &r)
{  int i=1;
   filein.open(filename, ios::binary);
   if (filein.fail ())
      return 0;
   else
   {  filein.read((char *) &rec, sizeof(rec));
      while (filein && rec.ID!=id)
      {  i++;
         filein.read((char *) &rec, sizeof(rec));
      }
      if (filein)
      {  r=rec;
         return i;
      }
      else   return 0;
   }
   filein.close();
}
```

Per la lettura dei dati si usa il metodo **fin.read()** inserito nella condizione di controllo del ciclo.

I vari campi dal file possono anche essere letti singolarmente, ad esempio, con:
rec.ID = fin.**get**();
for (int i=0; i<20; i++) rec.cognome[i]=fin.**get**();
.....

File binario di **struct**

Scrittura

Funzione per caricare nel file i dati relativi ad un record.

```
Inclusione librerie
#const char filename[10]="Dati";
typedef struct  { … } scheda;
void inserisci(int r);
scheda rec;
ifstream filein;
ofstream fileout;
int main()
{  int id;
   long pos;
   filein.open(filename, ios::in|ios::binary);
   if (filein.fail())    // se non esiste, apre in scrittura…
      fileout.open(filename, ios::out|ios::binary);
   else                  //… altrimenti apre in accodamento
      fileout.open(filename, ios::app|ios::binary);
  Inserisce record
}
```

File binario di **struct**

Scrittura

```
Inserisce record
    cin>>id;
    inserisci(id);              // aggiunge nuovo
    fileout.close();
    system("pause");
    return EXIT_SUCCESS;

void inserisci (int r)
{   rec.ID=r;
    cin.ignore(80,'\n');
    cout<<"\nCognome: ";
    cin.get(rec.cognome,20);
    cin.ignore(80,'\n');
    cout<<"\nTelefono: ";
    cin.get(rec.interno, 10);
    cin.ignore(80, '\n');
    fileout.write((char *) &rec, sizeof(scheda));
}
```

Notare l'utilizzo dei metodi:
- **cin.get()** per la lettura dei campi:
- **cin.ignore()** per eliminare il carattere '\n' lasciato dal metodo cin.get()
- **fout.write()** per scrivere il record nel file

Come per la lettura, possiamo anche scrivere:
fin.**put**(rec.codice);
for (int i=0; i<20; i++) fin.**put**(rec.cognome[i]);
….

File binario di **struct**

Eliminazione

```
Inclusione librerie
const char filename[10]="Dati";
typedef struct { …} scheda;
int ricerca(int id, scheda &r);
scheda rec;
ifstream filein;
ofstream fileout;
int main()
{  int id;
   long pos;
   filein.open(filename, ios::in|ios::binary);
   if (filein.fail())    cout<<"File inesistente!";
   else    Elimina record
   filein.close();
   system("pause");
   return EXIT_SUCCESS;
}
```

L'eliminazione richiede l'apertura in lettura e il controllo di esistenza del file.

File binario di **struct**

Eliminazione

Per cancellare si sfrutta la possibilità di **aprire il file in lettura e scrittura**.

Individuata la *posizione precedente* quella da cancellare, si pone il valore -1 al posto dell'ID (*cancellazione logica*) del record in memoria e si riscrive il record modificato.

```
Elimina record
    cout<<"ID.....:";
    cin>>id;
    filein.close();
    pos=ricerca(id, rec);              // trova posizione del record
    if (pos)                           // trovato - pos !=0
    {       fileout.open(filename, ios::in|ios::out);   // apre in I/O
            fileout.seekp((pos-1)*sizeof(scheda), ios::beg);
            // cerca la posizione precedente quella da eliminare
            rec.ID=-1;                 // cancella logicamante il record
            fileout.write((char *) &rec, sizeof(scheda));
                                       // riscrivendo il record cancellato
            fileout.close();
    }
```


File binario di **struct**

Stampa

Per stampare si apre il file in lettura e lo si scandisce record per record, stampando solo quando il campo ID è diverso da -1.

```
Inclusione librerie
int main()
{  ifstream file;
   file.open(filename, ios::binary);
   if (file.fail())      cout<<"File inesistente!";
   else {   file.read((char *) &rec, sizeof(rec));
            while (file)
            {   if (rec.ID!=-1) { cout<<"ID.....:"<<rec.ID<<endl;
                                  cout<<"Cognome.:"<<rec.cognome<<endl;
                                  cout<<"Interno.:"<<rec.interno<<endl;
                                }
                file.read((char *) &rec, sizeof(rec));
            }
        }
    file.close();
    system("pause");
    return EXIT_SUCCESS;
}
```

Cosa ho imparato

- File binari
- Operazioni su file binari
- Accesso diretto a record

Cosa ho imparato a fare

- Scrivere programmi facenti uso di file binari
- Utilizzare diversi tipi di accesso

Terminologia

- **cin.get**()
- **cout.get**()
- **ios::binary**
- **read**()
- **write**()
- **close**()
- **seekg**()
- **get**()
- **put**()

Altre fonti di informazione

- J. Purdum, C - ed. Jackson
- Romagnoli Ventura, C/C++ - Ed. Petrini
- A. Lorenzi et alii – Il linguaggio C++ - Ed. ATLAS
- A. Garavaglia, F.Petracchi, S.Forte
  Strutture dati e programmazione per oggetti, ed. Masson Scuola