



Accesso diretto per gli archivi con record a lunghezza costante

Nell'**accesso diretto** (in inglese, *random*) al file binario, le operazioni di lettura o scrittura devono essere precedute dall'operazione di posizionamento, che si realizza con i metodi **seekg()** in lettura (*seek get*) e **seekp()** in scrittura (*seek put*), specificando come argomento il numero di byte sul quale posizionarsi, relativamente all'inizio del file. La numerazione dei byte inizia da 0.

La variabile specificata come argomento dei metodi deve essere di tipo *long*.

Per esempio, per la lettura con accesso diretto, si usano le seguenti istruzioni:

```
nomestream.seekg(posizione);  
nomestream.read(variabile, sizeof(variabile));
```

I metodi **seekg()** e **seekp()** sono tipici dell'accesso random al file. Tali metodi hanno il compito di individuare la posizione di una specifica componente dello stream calcolando il numero di byte delle componenti che la precedono.

La sintassi completa è:

```
nomestream.seekg(posizione, inizio);
```

dove *posizione* (di tipo *long*) indica il numero di byte con i quali il puntatore al file viene spostato in avanti nel file, *inizio* indica il punto di partenza per il posizionamento e precisamente il secondo argomento può assumere uno dei seguenti valori:

- **ios::beg**, posizionamento a partire dall'inizio del file (valore di *default*);
- **ios::cur**, posizionamento a partire dalla posizione corrente del puntatore;
- **ios::end**, posizionamento a partire dalla fine del file.

Si ricordi che *seekg()* e *seekp()* si limitano ad individuare la posizione della componente, ma non eseguono operazioni di I/O che sono svolte dai metodi *read()* e *write()*.

È possibile conoscere in ogni istante la posizione corrente nello stream (di tipo *long*) attraverso i metodi **tellg()** in lettura e **tellp()** in scrittura.

Per esempio:

```
long posiz;  
posiz = nomestream.tellg();
```

Dopo aver creato le anagrafiche dei dipendenti con il programma del paragrafo precedente, è possibile fare la ricerca di un dipendente di cui si conosce la matricola, attraverso un **accesso diretto al file di record aventi lunghezza costante**.

ESEMPIO

Dato l'archivio anagrafico dei dipendenti di un'azienda, ricercare i dati di un dipendente corrispondente alla matricola fornita da tastiera.

All'inizio, dopo l'apertura dell'archivio, il programma conta il numero di record registrati. L'utente deve poi fornire il numero del record da ritrovare, che coincide con la matricola del dipendente: se fornisce un numero superiore al numero dei record registrati richiede di nuovo il numero di matricola. Per il posizionamento sul dipendente richiesto, si usa il metodo **seekg()**.

Si osservi l'utilizzo del metodo **clear()** per posizionare la lettura all'inizio del file:

```
archivio.clear();
```

PROGRAMMA C++

```
// RicDip.cpp: accesso diretto al file binario
//          (visualizzazione)
#include <iostream>
#include <fstream>
using namespace std;

struct Persona {
    int ID;           // matricola
    char nome[50];   // cognome e nome
    double stipendio; // stipendio
};

int main()
{
    Persona dipendente;
    int num = 0;
    ifstream archivio; //dichiara stream

    // apre in lettura
    archivio.open("anagrafe.dat", ios::in | ios::binary);
    if (!archivio) {
        cout << "Errore nell'apertura dell'archivio" << endl;
    }
    else {
        // calcola il numero di record registrati
        while (archivio.read((char *) &dipendente, sizeof(dipendente)))
            num++;
        cout << "Numero dei dipendenti registrati = " << num << endl;

        archivio.clear(); // si riposiziona all'inizio del file

        // chiede il numero del record
        long matr;
        do {
            cout << "Matricola del dipendente: ";
            cin >> matr;
        } while (matr < 1 || matr > num);
    }
}
```

```

    long posiz = (matr-1) * sizeof(dipendente);
    archivio.seekg(posiz);          // posizionamento nel file
    // legge il record
    archivio.read((char *) &dipendente, sizeof(dipendente));
    // visualizza il dipendente
    cout << dipendente.ID << ": "
          << dipendente.nome << '\t'
          << dipendente.stipendio << " euro"
          << endl;

    archivio.close();
}

return 0;
}

```

ESEMPIO

Dato l'archivio anagrafico dei dipendenti di un'azienda, aggiornare lo stipendio di un dipendente del quale viene fornita la matricola.

Il programma di **aggiornamento** (o di modifica) è in gran parte uguale all'esempio precedente per la ricerca con accesso diretto. Qui viene aggiunta la richiesta del valore per il nuovo stipendio e la riscrittura del record per aggiornare i dati del dipendente. L'operazione di **riscrittura** deve essere preceduta dal riposizionamento sul record del dipendente tramite il metodo **seekp()**, perché dopo una lettura ci si trova posizionati al record successivo. Nell'apertura del file sono state usate le due costanti **ios::in** e **ios::out**, perché il programma esegue operazioni sia di lettura sia di scrittura sullo stesso file.

PROGRAMMA C++

```

// AggioDip.cpp: accesso diretto al file binario
//                (aggiornamento)
#include <iostream>
#include <fstream>
using namespace std;

    struct Persona {
        int ID;           // matricola
        char nome[50];    // cognome e nome
        double stipendio; // stipendio
    };

int main()
{
    Persona dipendente;
    int num = 0;
    fstream archivio;    //dichiara stream

    // apre in lettura e scrittura
    archivio.open("anagrafe.dat", ios::in | ios::out | ios::binary);
    if (!archivio) {
        cout << "Errore nell'apertura dell'archivio" << endl;
    }
}

```

```

else {
    // calcola il numero di record registrati
    while (archivio.read((char *) &dipendente, sizeof(dipendente)))
        num++;
    cout << "Numero dei dipendenti registrati = " << num << endl;

    archivio.clear();           // si riposiziona all'inizio del file

    // chiede il numero del record
    long matr;
    do {
        cout << "Matricola del dipendente: ";
        cin >> matr;
    } while (matr < 1 || matr > num);

    long posiz = (matr-1) * sizeof(dipendente);
    archivio.seekg(posiz);     // posizionamento sul record
    // legge il record
    archivio.read((char *) &dipendente, sizeof(dipendente));
    // visualizza il dipendente
    cout << dipendente.ID << ": "
         << dipendente.nome << '\t'
         << dipendente.stipendio << " euro"
         << endl;

    // nuovo stipendio
    cout << "Nuovo stipendio (euro): ";
    cin >> dipendente.stipendio;

    // si riposiziona e riscrive il record
    archivio.seekp(posiz);
    archivio.write((char *) &dipendente, sizeof(dipendente));

    archivio.close();
}

return 0;
}

```

Per implementare l'operazione di **cancellazione di un record**, occorre eliminare il contenuto dei campi oppure marcare con un carattere (per esempio *) il contenuto del primo campo, in modo che in fase di lettura questi record vengano trascurati. Tuttavia il record non viene fisicamente eliminato dal file. Quindi per evitare lo spreco di spazio dovuto alla presenza di record vuoti è opportuno eseguire un procedimento di cancellazione dei record secondo questi passi:

- si crea un nuovo file con la stessa struttura di record
- leggendo il file di partenza in modo sequenziale a partire dal primo record, si copiano i record che devono essere mantenuti nel nuovo file
- alla fine si chiudono entrambi i file e si cancella il file originale
- si rinomina il nuovo file assegnando ad esso il nome del file eliminato.

Si osservi però che, così facendo, si perde la corrispondenza tra posizione del record nel file e valore del campo *ID*.