Operazioni di Input/Output in C++

prof. Riccardo Torlone
Università di Roma Tre

Operazioni di lettura e scrittura

- Un programma comunica con l'esterno attraverso strutture logiche dette **flussi** o **stream**
- I flussi vengono associati:
  - o a dispositivi fisici (video, tastiera, stampanti),
  - o a file residenti in memoria secondaria.
- Le operazioni più comuni che si effettuano sui flussi sono quelle di lettura e scrittura che possono essere:
  - *formattate*: prima del trasferimento convertono il contenuto di un flusso da una rappresentazione interna a dati tipati (e viceversa);
  - *non formattate*: non effettuano conversioni prima del trasferimento;
- 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* (o casuale): si può operare su un dato qualunque del flusso

Input/Output in C++

- In C++ esistono alcune classi predefinite che gestiscono le operazioni di I/O
- Queste classi contengono, nella parte pubblica, le operazioni disponibili su un flusso e, nella parte privata, una struttura, detta **buffer**, che memorizza blocchi di caratteri in transito tra la memoria principale e il dispositivo (o il file) associato al flusso
- Un flusso è un oggetto di una di queste classi predefinite ed è costituito (logicamente) da una sequenza di caratteri terminata da un carattere speciale (EOF)

Classi per la gestione dell'I/O

- Le classi predefinite per la gestione dell'I/O sono tutte derivate dalla classe base **IOS** che definisce il concetto astratto di flusso e tutte le operazioni di base su di esso
  - Per le operazioni di I/O che coinvolgono dispositivi fisici si fa uso delle classi **istream** (ingresso) e **ostream** (uscita) che sono definite nella libreria *iostream.h*
  - Per le operazioni di I/O che coinvolgono file si fa uso delle classi **ifstream** (ingresso) e **ofstream** (uscita) e **fstream** (ingresso/uscita) che sono definite nella libreria *fstream.h*
- Per le proprietà della derivazione, si usano, in entrambi i casi, le stesse operazioni (definite nella classe IOS), opportunamente ridefinite

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 formattati di base sono >> (ingresso) e << (uscita) che sono definiti per i tipi fondamentali, per i puntatori e per le stringhe
- Tali operatori restituiscono un riferimento a un flusso e possono essere quindi composti
- L'ordine di precedenza di tali operatori è più alto solo degli operatori logici

Esempio

```
int x, y;
cin >> x >> y;
cout << “x+y=“ << x+y << endl;
        // endl inserisce il carattere “\n” e vuota il buffer
cout << “x < y=“ << (x<y);
```

Nota: L’operatore **>>** scarta tutti gli spazi, le tabulazioni e il carattere di fine riga inseriti in ingresso

Altre operazioni di ingresso

Oltre all'operatore di base >>, è possibile far uso di altre funzioni di ingresso non formattate:

- **int get()**: legge il prossimo carattere (qualunque sia) dal flusso di ingresso e lo restituisce;
- **istream& get(char& ch)**: legge il prossimo carattere (qualunque sia) dal flusso di ingresso e lo memorizza in ch;
- **istream& getline(char& st,int n,char l='\n')** legge una stringa di caratteri dal flusso di ingresso e la memorizza in st; vengono letti n caratteri oppure finché non si raggiunge il carattere l;
- **int gcount()**: restituisce il numero di caratteri effettivamente letti dall'ultima getline;
- **istream& read(char* ps, int n)**: legge una sequenza di caratteri lunga n dal flusso di ingresso e la memorizza a partire dall'indirizzo ps.

Esempio

```
...
int i;
char c;
char str[100];
i = cin.get(); // converte in un intero il carattere letto in ingresso
cout            <<“Letto il carattere n.=“<<i+1-’a’;
i=0; cin.get(c);
while ( c  != '\n' ) {
      i++; cin.get(c);
}
cout            << “Numero di caratteri letti=“ << i;
cin.getline(str,100,’.’);
cout            << “Numero di caratteri letti=“  << cin.gcout();
...
```

Altre operazioni di uscita

Oltre all'operatore di base <<, è possibile far uso di altre funzioni di uscita non formattate:

- **ostream& put(char ch)**: scrive il carattere ch (qualunque sia) nel flusso di uscita;
- **ostream& write(const char* ps, int n)**: scrive n caratteri della stringa puntata da ps sul flusso di uscita

Esempio

```
#include <iostream.h>
main() {
    int i=0;
    char c;
    char str[100];
    while ( (c = cin.get()) != '\n' ) {
      cout.put(c); i++;
    }
    cout << “Numero di caratteri scritti =“ << i << endl;
    cin.read(str,10);
    cout << “Primi 5 caratteri della “ << “stringa letta:” << endl;
    cout.write(str,5);
}
```

Controllo sui flussi

Ad ogni flusso sono associati alcuni indicatori di stato (campi privati della classe IOS) che di default valgono 0:

- **fail**: viene posto a 1 quando si verifica un errore recuperabile, che causa il fallimento di una operazione di I/O ma ne consente altre dopo un'operazione di ripristino (per esempio quando ci si aspetta un intero e si legge un carattere o l'EOF);
- **bad**: viene posto a 1 quando si verifica un errore non recuperabile, che impedisce altre operazione di I/O sul flusso (per esempio quando non è possibile aprire un flusso);
- **eof**: viene posto a 1 quando si incontra l'EOF

Funzioni di controllo

Per fare controlli sui flussi è possibile usare le funzioni:

- **void clear(int i=0)**: modifica i valori degli indicatori di stato e ripristina il flusso;
- **int fail()**: restituisce il valore di fail;
- **int bad()**: restituisce il valore di bad;
- **int eof()**: restituisce il valore di eof;
- **int good()**: restituisce 1 se tutti gli indicatori sono a 0.

Nota: l'operatore >> restituisce 1 se fail e bad sono a 0.

Esempio

```
...
int i; char c;
i = cin.get();
if (cin.fail())
    cout >> “Errore di lettura”;
cin.clear();
while(!cin.eof()){
    cin.get(c); cout.put(c);
}
while (cin >> c) cout << c;
```

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, tra cui:

- **dec**, **hex**, **oct**: selezionano una base di numerazione;
- **setw()**: stabilisce l'ampiezza del campo di stampa;
- **setprecision()**: stabilisce il numero di cifre significative per i numeri reali.

```
int i; char c;
cin >> i;
cout << hex << i << ' '  << dec << i << endl;
cout << setw(10) << i << endl;
cout << setprecision(9) << i/7 << endl;
```

I manipolatori setw() e setprecision() sono definiti nella libreria iomanip.h.

Indicatori di formato

Il formato di uscita può essere controllato anche attraverso la modifica di opportuni indicatori di formato associati a ogni flusso

```
cout.fill('#'); // ha effetto fino alla prossima istruzione fill
int a=1, b=2, c=3;
cout.width(5); // ha effetto solo sulla successiva istruzione
cout << a << endl; // stampa ####1
cout.width(8);
cout << b << endl; // stampa #######2
cout << c << endl; // stampa 3
```

Input/Output su file

- Per le operazioni di ingresso/uscita da/verso file è sufficente 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::in|ios::out (apertura in lettura/ scrittura).
- Questi dati si possono specificare anche in fase di dichiarazione.
- Per default, un oggetto di ifstream viene aperto in lettura mentre un oggetto di ofstream viene aperto in scrittura.

Esempio

```
ifstream infile;
ofstream outfile;
fstream suofile;
fstream altrofile(“prova.dat”,ios::out);
....
infile.open(“uno.txt”);
outfile.open(“due.txt”,ios::out);
suofile.open(“tre.txt”,ios::in|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.h>
void main()  {
        ofstream outfile; // denota il file sul quale scrivere i dati
        char c;
        outfile.open("dati.txt");
        if (!outfile.good()) {
                cout      << "Errore in apertura“ << “ del file dati” << endl;
                return;
        }
        while((c = cin.get()) != '\n') outfile.put(c);
        outfile.close();
}
```

Copia di file

```
#include <fstream.h>
void main()  {
        fstream infile; // file di ingresso
        fstream outfile; // file di uscita
        infile.open("dati.txt",ios::in);
        if (infile.bad()) {
                cout << "Errore in apertura“ << “ del file dati” << endl;
                return;
        }
        outfile.open("copia.txt",ios::out);
        if (outfile.bad()) {
                cout << "Errore in apertura“ << “ del file copia” << endl;
                return;
        }
```

Copia di file (continua)

```
        char c;
        infile.get(c);
        while(!infile.eof()){
                if (infile.fail()) {
                        cout<<“Errore in lettura”<<endl;
                        infile.clear();
                }
                else outfile << c;
                infile.get(c);
        }
        outfile.close();
        infile.close();
}
```

Accesso diretto a file

E’ possibile accedere ad un file anche in maniera diretta (casuale) utilizzando una delle seguenti istruzioni:

- **istream& seekg(long i)**: si posiziona sul carattere i del file per una successiva operazione di lettura (posizionamento assoluto);
- **istream& seekg(long i, seek_dir s)**: si posiziona sul carattere i del file a partire dalla posizione s per una successiva operazione di lettura (posizionamento relativo);
- **ostream& seekp(long i)**: si posiziona sul carattere i del file per una successiva operazione di scrittura (posizionamento assoluto);
- **ostream& seekp(long i, seek_dir s)**: si posiziona sul carattere i del file a partire dalla posizione s per una successiva operazione di scrittura (posizionamento relativo);
- **long tellg()**: restituisce la posizione corrente;
- **long tellp()**: restituisce la posizione corrente.

Il parametro di tipo seek_dir

Il parametro di tipo seek_dir può assumere uno dei seguenti valori:

- **ios::beg** indica l'inizio del file;
- **ios::cur** indica la posizione corrente;
- **ios::end** indica la fine del file.

Programma che stampa il contenuto di un file

```
#include<fstream.h>
void main()  {
        fstream f;
        char nomef[40];
        cout << “Nome del file: “;
        cin >> nomef;
        f.open(nomef, ios::in | ios::binary);
        if (!f) {
                cout << "Errore in apertura" << endl;
                exit(1);
        }
```

Programma che stampa il contenuto di un file (II)

```
        long n, nchar=0; char ch;
        f.seekg(0,ios::end);
        nchar=f.tellg(); // lunghezza del file
        cout    << “Il file contiene ” << nchar << “ caratteri” << endl;
        cout    << “Posizione iniziale:”; cin >> n;
        if (n<0 || n>nchar) {
                cout << “Posizione errata” << endl;
                exit(1);
        }
        f.seekg(n,ios::beg);
        while(!f.eof()) { f.get(ch); cout.put(ch); }
        f.close();
}
```

Gestione di un dizionario in memoria secondaria

```
#include <fstream.h>
typedef int tipodato;
typedef int tipochiave;
class dizionario {
public:
        dizionario()  { dim=sizeof(record); status=0; }
        void crea()  { svuota(); }
        tipodato trovainfo(tipochiave);
        void inserisci(tipochiave,tipodato);
        void modifica(tipochiave,tipodato);
        void cancella(tipochiave);
        void stampa();
        int stato()  { return status; };
```

Classe dizionario (continua)

```
private:
        fstream f;
        struct {
                int flag;
                tipochiave chiave;
                tipodato dato;
        } record;
        int dim;
        int status;
        void svuota();
        boolean cerca(tipochiave,int &);
};
```

Classe dizionario (continua)

```
void dizionario::svuota() {
    f.open("out.txt", ios::out);
    f.close();
};
boolean dizionario::cerca(tipochiave k, int &i)  {
    i=0; boolean trovato = falso;
    f.open("out.txt", ios::in);
    f.read((char*)&record,dim);
    while (!f.eof() && trovato==falso)  {
        if (record.chiave==k && record.flag==0) trovato=vero;
        else  { f.read((char*)&record,dim); i++; }
    }
    f.close(); return trovato;
};
```

Classe dizionario (continua)

```
tipodato dizionario::trovainfo(tipochiave k) {
        tipodato x=0;
        int i=0;
        boolean esiste=cerca(k,i);
        if (esiste)  {
                f.open("out.txt", ios::in);
                f.seekg(i*dim,ios::beg);
                f.read((char*)&record,dim);
                x=record.dato;
                status=0; f.close();
        }
        else status=1;
        return x;
}
```

Classe dizionario (continua)

```
void dizionario::inserisci(tipochiave k,tipodato d) {
        int i=0;
        boolean esiste=cerca(k,i);
        if (esiste) status=1;
        else {
                f.open("out.txt", ios::out|ios::app);
                record.flag=0;
                record.chiave=k;
                record.dato=d;
                f.write((char*)&record,dim);
                status=0;
                f.close();
        }
};
```

Classe dizionario (continua)

```
void dizionario::cancella(tipochiave k) {
        int i=0;
        boolean esiste=cerca(k,i);
        if (!esiste) status=1;
        else {
                f.open("out.txt", ios::in|ios::out);
                f.seekg(i*dim,ios::beg);
                record.flag=1;
                f.write((char*)&record,dim);
                status=0;
                f.close();
        }
};
```

Classe dizionario (continua)

```
void dizionario::stampa() {
        cout << "Contenuto Dizionario:" << endl;
        f.open("out.txt", ios::in);
        f.read((char*)&record,dim);
        while (!f.eof())  {
                if (record.flag==0)
                cout      << "chiave:" <<record.chiave<< " "
                                << "dato:" << record.dato << endl;
                f.read((char*)&record,dim);
        }
        f.close();
};
```

Uso della classe dizionario

```
void main()  {
    dizionario d; d.crea();
    d.inserisci(5,7); if (d.stato()==0) cout << "Operazione eseguita" << endl;
    d.inserisci(2,8); if (d.stato()==0) cout << "Operazione eseguita" << endl;
    d.inserisci(8,4); if (d.stato()==0) cout << "Operazione eseguita" << endl;
    d.inserisci(2,4); if (d.stato()==0) cout << "Operazione eseguita" << endl;
    d.inserisci(12,4); if (d.stato()==0) cout << "Operazione eseguita" << endl;
    d.inserisci(7,11); if (d.stato()==2) cout << "Overflow" << endl;
    d.stampa(); cout << "RICERCHE" << endl;
    d.cancella(2);
    tipodato a=d.trovainfo(8);
    if (d.stato()==0)  cout << "chiave:" << 8  << " dato:" << a << endl;
    d.cancella(12); d.stampa();
}
```