# Fondamenti di Informatica II

## 24. Gestione I/O e File in C++


Corso di Laurea in Ingegneria Informatica
A.A. 2008-2009
2° Semestre – Corso (A-M)
Prof. Giovanni Pascoschi


## Gli stream in C++

- **stream** → è un'astrazione che rappresenta un flusso di dati che scorre tra una "**sorgente**" e una "**destinazione**" → idea di Stroustrup : una sequenza di caratteri
- esempi:
  - interattivi (iostream)
    - **cin** → stream di input associato con la tastiera
    - **cout** → stream di output associato con il display
  - file
    - **ifstream** → definisce un nuovo stream di input (associato normalente ad un file)
    - **ofstream** → definisce un nuovo stream di output (normalmente associato ad un file)



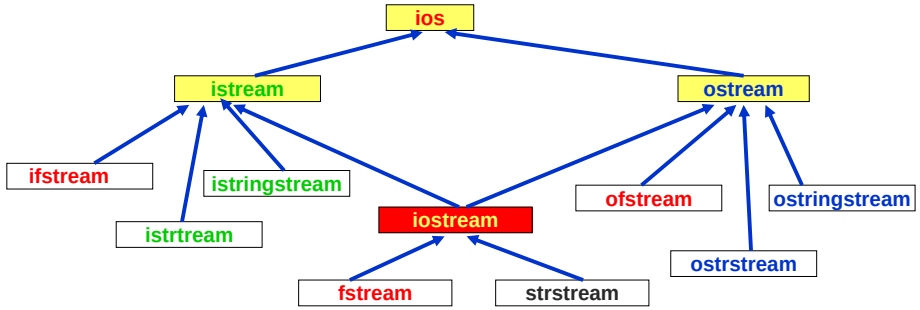
## Gli stream in C++

- la prima classe storicamente implementata da Schwarz è **istream** che identifica un flusso di caratteri da un dispositivo di input arbitrario a un applicativo software arbitrario

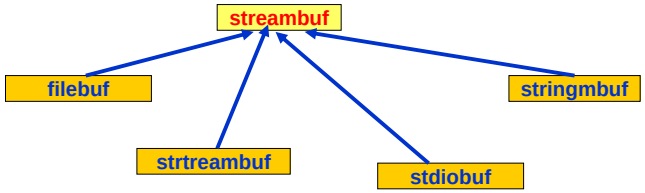
1. la classe base è **ios** per il processamento ad **alto livello**
2. la classe base è **streambuf** per il processamento a **basso livello** (dispositivi fisici)



## Gerarchia delle classi in C++ (ios)



## La gerarchia delle classi in C++ (streambuf)




- streambuf→ per gestire un'interfaccia ai dispositivi fisici (processamento a basso livello)
- stdiobuf→ si utilizza per combinare I/O di flussi C++ con le funzioni C-style



## Header files per la libreria degli stream I/O

- iostream→ contiene le informazioni di base richieste per tutte le operazioni di stream bidirezionali di I/O
- iomanip→ contiene le informazioni usate per ottenere I/O formattati con manipolatori di flusso stream parametrizzati
- fstream → contiene le informazioni per gestire le operazioni di I/O su file
- strstream→ contiene informazioni per gestire le operazioni di I/O in memoria (p.e. per le stringhe in memoria)



## Flussi standard

- il flusso cin definito nella classe istream è associato all'input standard (di solito se non è reindirizzato coincide con la tastiera (stdin))
- il flusso cout definito nella classe ostream è associato all'output standard (di solito se non è reindirizzato coincide con lo schermo (stdout))
- il flusso cerr definito nella classe ostream è associato al flusso standard di errore (di solito diretto verso lo schermo) e non usa un buffer di memoria
- il flusso clog è simile a cerr ma usa un buffer di memoria



## Flussi I/O su file

- il flusso ifstream derivata dalla classe istream è utilizzata per gestire la lettura da file (solo lettura)
- il flusso ofstream derivata dalla classe ostream è utilizzata per gestire la scrittura su file (solo scrittura)
- il flusso fstream derivata da iostream è utilizzata per gestire sia la lettura che la scrittura da file (lettura+scrittura contemporaneamente)

## La classe istream

- la classe istream deriva dalla classe ios
- la classe istream consente di gestire un flusso di input (metodi per acquisire dati in modalità formattata e non formattata)
- al fine di consentire operazioni di I/O ad alto livello, l'operatore ">>" è usato in overloading con tutti i tipi predefiniti del C++
- l'operatore >> si utilizza applicandolo a un oggetto della classe istream (cin):

```
cin >> a;      // ">>" cerca di estrarre dall'oggetto cin un flusso di caratteri
```



## La classe istream (cin)

- l'utilizzo di cin consente di monitorare lo stato di un eventuale errore in fase di acquisizione (p.e. acquisizione di un carattere quando invece si attende un intero)
- la classe istream ha un attributo che indica lo stato (flag = good/bad/fail)
- cin.good( ) → controlla se la cin è stata effettuata correttamente
  ritorna valore non-zero ("true") se lo stream non ha incontrato dei problemi (come lettura dell'EOF o file non esistente)
- cin.bad( ) → controlla se la cin è stata effettuata con qualche errore
  ritorna valore non-zero ("true") se lo stream è totalmente inutilizzabile, come ad esempio se il file non si puo' aprire
- cin.fail( ) → controlla se la cin non si è potuta completare
  ritorna valore non-zero ("true") se l'ultimo comando cin è fallito
- cin.clear( ) → serve per ripristinare un flag (eofbit) che permette di individuare se l'input è terminato oppure no (ctrl D o ctrl Z per segnalare la EOF)



## La classe istream (get & getline)

- siccome la cin interrompe l'acquisizione di una variabile allorquando si incontra uno spazio, è necessario usare la cin.get( ) e la cin.getline( ) per acquisire stringhe con spazi vuoti
- la get( ) serve per acquisire un solo carattere o piu' caratteri contemporaneamente:

```
istream& get( ); //prototipo del metodo get senza parametri
istream&  get(char& c); // prototipo del metodo get con passaggio di un parametro
istream&    get(char* buffer, int n, char sep='\n'); // prototipo del metodo get simile a
                                                     getline (3° campo opzionale)
```

istream& è il riferimento all'oggetto flusso di cui è membro

- la getline( ) inserisce automaticamente il carattere di terminazione '\0' come ultimo carattere della stringa

sintassi

```
cin.getline (buffer, 20, char sep='\n'); // simile alla sintassi della get
```

- la differenza tra get e getline è che quest'ultima estrae il carattere separatore dal flusso di input prima di aggiungere il carattere '\0' nella stringa



## La classe istream (getline)

- la getline puo' creare dei problemi nell'acquisizione:

```
int main( ){
      string regione;
      string citta;
      cin.getline(regione);
      cin.getline(citta);
      .....
}
```

- se si inserisce prima puglia e poi bari le variabili acquisite sono:

regione = puglia
citta = '\n' '\0'
(il \n rimane nel buffer a fronte della prima getline. La successiva getline() acquisisce proprio il \n poichè per default è il carattere di separazione→si ferma ed aggiunge \0)

- per ovviare al problema ci sono 2 possibilità:

```
cin.getline(temp, 2); // x pulire il buffer della tastiera in una variabile di appoggio
cin.ignore(80, '\n'); // rimuove dal buffer di input tutti i caratteri fino a \n
```

## La classe ostream

- deriva dalla classe ios
- la classe ostream consente di gestire un flusso di output (metodi per mettere in uscita dati in modalità formattata e non formattata)
- per consentire operazioni di I/O ad alto livello l'operatore "<<" è usato in overloading con tutti i tipi predefiniti del C++
- l'operatore << si utilizza applicandolo a un oggetto della classe ostream (cout):

```
cout << a;      // "<<" cerca di porre in uscita verso l'oggetto cout un flusso di caratteri
```



## Overloading degli operatori "<<" e ">>" (esempio 2)

```
class Complesso {
    double a;
    double b;
public:
    Complesso(double a = 0, double b = 0);
    void setReale(double);
    void setImmag(double );
    double getReale( ) const;
    double getImmag( ) const;
    Complesso operator+(const Complesso &n) const;
    Complesso operator-(const Complesso &n) const;
    Complesso operator*(const Complesso &n) const;
    Complesso &operator=(const Complesso &n);
    bool operator==(const Complesso &n) const;
    bool operator!=(const Complesso &n) const;
    friend ostream& operator<<(ostream &output, const Complesso &n);
    friend istream& operator>>(istream &input, Complesso &n);
};
```



## Overloading degli operatori "<<" e ">>"(esempio 2)

```
// operatore di assegnamento
Complesso& Complesso::operator=(const Complesso &n) {
    a = n.a;
    b = n.b;
    return *this;
}

// operatore di uguaglianza
    bool Complesso::operator==(const Complesso &n) const {
    return a == n.a && b == n.b;
}

// operatore di disuguaglianza
    bool Complesso::operator!=(const Complesso &n) const {
    return a != n.a || b != n.b;
} etc etc etc
```



## Overloading degli operatori "<<" e ">>" (esempio 2)

```
// operatore di inserimento
ostream &operator<<(ostream &output, const Complesso &n) {
    output << n.a;
    if (n.b >= 0) {
        output << "+j" << n.b;
        }
   else {
       output << "- j" << -1*n.b;
       }
   return output;
}

// operatore di estrazione
   istream &operator>>(istream &input, Complesso &n) {
   input >> n.a >> n.b;
   return input;
}
```

## Overloading degli operatori “<<“ e “>>”(esempio 2)

```
int main( ) {
   Complesso x;
   Complesso z;
   double re, imm;
   cout << "Valore iniziale di x = " << x << endl;          // 0+j0
   cout << "Immetti i nuovi coefficienti di x: ";
   cin >> x;
   // cin >> re >> imm;
   // x.setReale(re);
   // x.setImmag(imm);
   cout << “Nuovo valore di x=“ << x << endl;
   // cout << "Nuovo valore di x = " << x.getReale( ) << " + j"  << x.getImmag( ) << endl;
   .....
   system ("pause");
}
```



## I manipolatori dell’output

- i manipolatori consentono di operare sugli oggetti di tipo stream

```
#include <iomanip>
```

- endl è un manipolatore che invia ‘\n’ allo stream di output e svuota il buffer di I/O
- cout.flush( ) → consente lo svuotamento del buffer associato allo stream
- dec /oct / hex / setw / setfill / setprecision (ved. 1° modulo)



## Gli indicatori di formato

- esistono dei manipolatori specializzati chiamati indicatori di formato che utilizzano dei flag
- i flag si possono settare tramite il metodo setiosflags e resettare tramite resetiosflags
- è possibile combinare diversi flags tramite un OR sui bit ( | )



## Gli indicatori di formato

| | |
|---|---|
| ios::left | allinea a sinistra |
| ios::right | allinea a destra |
| ios::dec | rappresenta in base 10 |
| ios::oct | rapresenta in base 8 |
| ios::hex | rappresenta in base 16 |
| ios::scientific | rappresenta nel formato esponenziale |
| ios::fixed | rappresenta i numeri float con un numero fisso di decimali |

- esempio:

```
cout    << 10.
        << setiosflags(ios::scientific)
        << 10.
        << endl;
```

## I/O da file in C++

- il file è inteso come flusso esterno di dati

- tre classi per gestire i file (ifstream, ofstream, fstream) contenute nell'header file <fstream>

- apertura implicita del file

- ifstream fin ("dati");
- ofstream fout ("dati", ios::out);
- fstream finout; // si puo' aprire sia in lettura che in scrittura



## Modi di apertura del file

| Nome | Descrizione |
|---|---|
| ios::in | Apre il file in lettura |
| ios::out | Apre il file in scrittura |
| ios::app | Apre il file in scrittura in append |
| ios::ate | Apre il file →cerca la fine del file |
| ios::trunc | cancella tutto il contenuto precedente nel file |
| ios::nocreate | se il file non esiste, l'apertura del file è impossibile |
| ios::noreplace | se il file esiste, l'apertura del file ritorna un errore |
| ios::binary | apre il file in modalità binaria |



## Classe fstream

- apertura esplicita del file

- fstream f
- f.open ("Archivio.dat", ios::in | ios:: out | ios:: binary)

- il metodo open di solito non si usa perchè le tre classi di gestione dei file hanno costruttori che aprono automaticamente il file

- quindi la sintassi che si puo' usare è quella che consente l'apertura implicita:
- fstream f ("Archivio.dat", ios::in | ios:: out | ios:: binary); // apre il file

- f.close( ); // metodo per chiudere il file



## Errori sui file

- 1° modalità :

if (!nomestream) cout << "errore nell'apertura del file"<< endl;

- 2° modalità :

if(nomestream.fail( )) cout<< "errore nell'apertura del file"<<endl;

- 3° modalità :
- tramite la gestione delle eccezioni try...catch

## Classi per i file

- le classi per i file ereditano i metodi già visti per iostream:
  - int good( );
  - int fail( );
  - int eof( );
- la lettura e scrittura su file di testo viene effettuato tramite gli operatori << e >> e le operazioni già previste per cin e cout (get, getline, ecc.)



## Esempio di scrittura su file I/O di testo

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

int main( ) {
        ofstream fout;
        fout.open("registro.dat");
        fout << "Hello World";
        fout.close( );
        return 0;
}
```



## Esempio di lettura da file I/O di testo

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

int main( ) {
        ifstream fin;
        fin.open("archivio.txt");
        string line;
        while(!fin.eof( )) {
            getline(fin,line);
            cout<<line;
            }
        fin.close( );
        return 0;
}
```



## Gestione dei file binari

- get( ) → legge un byte e punta al prossimo byte
- put( ) → scrive un byte e punta alla prossima locazione
- read( ) → legge un blocco e punta al prossimo blocco
- write( ) → scrive un dato e punta al prossimo blocco

## Gestione dei file binari

- Sintassi:
- nomestream.write((char*) &dipendente, sizeof(dipendente));
- nomestream.read((char*) & dipendente, sizeof(dipendente));

N.B.: viene fatto un casting sul record di dipendente, perchè la funzione deve ricevere un puntatore a carattere



## Gestione dei file binari

- Esistono due puntatori (in lettura e scrittura)
- seekg( ) → vai ad una specifica posizione in lettura (current get position)
- seekp( ) → vai ad una specifica posizione in scrittura (current put position)
- sintassi della seek:

nomestream.seekg(posizione, inizio)

  - dove posizione è long e inizio puo' assumere i seguenti valori:
    - ios::beg
    - ios::cur
    - ios::end



## Gestione dei file binari

- tellg( ) → ritorna un intero che indica la posizione corrente del puntatore (in modalità lettura)
- tellp( ) → ritorna un intero che indica la posizione corrente del puntatore (in modalità scrittura)
- n.b.: ogni volta che si fa un'operazione di I/O il puntatore si sposta automaticamente



## Esempio Accesso Sequenziale (1)

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

struct persona {
        int matricola;
        string cognome;
        string nome;
        int livello;
};
typedef struct persona dip;
dip dipendente;
fstream fs;

void registra(void);
void visualizza(void);
```

## Esempio Accesso Sequenziale (2)

```
int main( ) {
        // registrazione
        fs.open("registro.dat",ios::in | ios::binary);
        registra( );
        fs.close();
        // visualizzazione
        fs.fopen("registro.dat", ios::out | ios::binary);
        if( fs.fail( ) != FALSE) {
                visualizza( );
                fs.close( );
                }
        else cout<<"Errore: l'archivio non esiste"<<endl;
        return 0;
}
```



## Esempio Accesso Sequenziale (3)

```
void registra() {
        int risposta;
        do {
                cout<<"Inserimento di un dipendente"<<endl;
                cout<<"Matricola:"<<endl;
                cin>>dipendente.matricola;
                cout<<"Cognome:"<<endl;
                cin>>dipendente.cognome;
                cout<<"Nome:"<<endl;
                cin>>dipendente.nome;
                cout<<"Livello:"<<endl;
                cin>>dipendente.livello;
                fs.write((char *)&dipendente, sizeof(dip));
                cout<<"Elenco finito? (0 = no, 1 = si):"<<endl);
                cin>>risposta;
         } while(!risposta);
        return;
}
```



## Esempio Accesso Sequenziale (4)

```
void visualizza() {
        cout<<"Elenco dipendenti"<<endl;
        fs.read((char *)&dipendente, sizeof(dip));
        while(fs.feof(fp)==FALSE) {
                cout<<"Matricola:"<<endl;
                cout<<dipendente.matricola;
                cout<<"Cognome:"<<endl;
                cout<<dipendente.cognome;
                cout<<"Nome:"<<endl;
                cout<<dipendente.nome;
                cout<<"Livello:"<<endl;
                cout<<dipendente.livello;
                fs.read((char *)&dipendente, sizeof(dip));
         }
        return;
}
```



## Esempio Accesso Diretto (1)

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

struct persona {
        int matricola;
        string cognome;
        string nome;
        int livello;
};
typedef struct persona dip;
dip dipendente;
fstream fs;

void registra(void);
```

## Esempio Accesso Diretto (2)

```
int main( ) {
        // registrazione
        fs.open("registro.dat", ios::in | ios::binary);
        if( fs.fail( ) != FALSE) {
                registra( );
                fs.close( );
                }
        else cout<<"Errore: l'archivio non esiste"<<endl;
        return 0;
}
```



## Esempio Accesso Diretto (3)

```
void registra() {
        int i;
        long posizione;
        cout<<"Inserimento di un dipendente"<<endl;
        cout<<"inserire indice record dove inserire il dipendente"<<endl;
        cin>>i;
        while(i > 0) {
            posizione = (i – 1) * sizeof(dip);
            fs.seekg(posizione, ios::beg);
            if( fs.read((char *)&dipendente, sizeof(dip)) && dipendente.cognome[0]!='\0') {
                cout<<"Attenzione codice gia' presente"<<endl;
                cout<<"Matricola:"<<endl;
                cin>>dipendente.matricola;
                cout<<"Cognome:"<<endl;
                cin>>dipendente.cognome;
                cout<<"Impossibile registrazione"<<endl;
                }
```



## Esempio Accesso Diretto (4)

```
            else {
                cout<<"Matricola:"<<endl;
                cin>>dipendente.matricola;
                cout<<"Cognome:"<<endl;
                cin>>dipendente.cognome;
                cout<<"Nome:"<<endl;
                cin>>dipendente.nome;
                cout<<"Livello:"<<endl;
                cin>>dipendente.livello;
                fs.seeks(posizione, ios::beg);
                fs.write((char *)&dipendente, sizeof(dip));
                 }
        cout<<"inserire indice record dove inserire il dipendente (0 termina)"<<endl;
        cin>>"i";
        }
        fs.close( );
        return;
}
```



## Output su stampante

- #include <fstream>
- ofstream dispositivo (nome_disp);

```
#include <fstream>
using namespace std;
int main( ) [
    char titolo[50];
    char autore[20];
    cout << "Inserire titolo";
    cin >> titolo;
    cout<< "Inserire autore";
    cin >> autore;
    ofstream stampante ("LPT1");
    stampante << "il titolo e' " << titolo;
    stampante << "l'autore e' " << autore;
return 0
]
```

## Riepilogo della lezione

### Gestione dell'I/O e dei file in C++

- Gestione dell'I/O in C++
- Gerarchia delle classi di I/O
- Classi istream/ostream
- Gestione dei file in C++ (accesso sequenziale e diretto)
- Output su stampante



## Fine della lezione

- Domande?