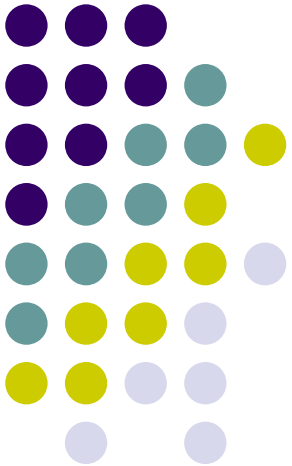
Università di Napoli Federico II - Dip. di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea in Ingegneria Informatica

DIE TI

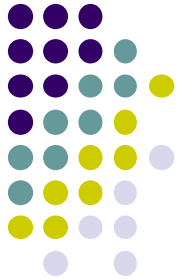
Corso di Programmazione

Richiami

Operazioni su File (Testo)

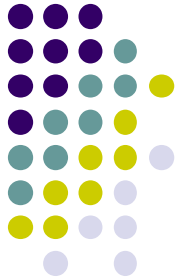


Il concetto di “Stream”



- Il sistema di I/O del C e del C++ si basa sul concetto di canale (stream) inteso come mezzo attraverso cui fluiscono le informazioni provenienti o inviate dai diversi dispositivi hardware (le unità esterne)
- In altre parole **lo stream è un flusso di dati** (sorgente o destinazione) che può essere associato ad un disco o ad altre periferiche
- Questi canali possono essere connessi a dispositivi fisici diversi mantenendo inalterato il proprio comportamento: un flusso di informazioni, per mezzo delle stesse modalità, può essere mandato in uscita su un video, su una stampante o su un file ed analogamente un flusso di informazioni può provenire adottando le stesse modalità dalla tastiera o da un file presente nella memoria di massa

Stream di dati



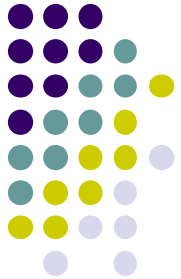
In altre parole uno stream è una astrazione che consente di definire una interfaccia comune a diversi dispositivi di I/O in modo da:

- rendere la scrittura dei programmi indipendenti dal particolare dispositivo impiegato;

- semplificare i problemi di portabilità dei programmi stessi.

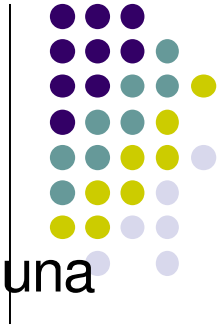
Lo stream consente di definire un'interfaccia semplice (astrazione di un dispositivo generico) e uniforme verso l'utente.

Stream di testo e stream binari



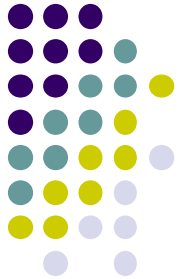
- La libreria (sia quella C che quella C++) supporta sia stream di testo che stream binari
- Uno stream di testo è una sequenza di linee, ciascuna **formata da caratteri** e chiusa dal carattere ‘\n’
- Uno stream binario è una *sequenza di byte* che registrano dati interni (di qualsiasi tipo) chiusa da un terminatore **end-of-file** garantendo che esista corrispondenza tra ciò che viene scritto e successivamente letto sullo stesso sistema

Connessione di uno stream ad un dispositivo



- Uno stream viene connesso ad un dispositivo di I/O tramite una operazione di apertura.
  - In particolare, quando *un file viene aperto*, è creato un oggetto ed uno stream viene associato all'oggetto
- Gli stream forniscono un canale di comunicazione tra un programma ed uno specifico file o dispositivo cui sono associati
- La connessione viene interrotta con una operazione di chiusura
- La ricezione di dati da un dispositivo di input è detta "estrazione da un flusso"
- La trasmissione di dati ad un dispositivo di uscita è derra "inserimento in un flusso"

Operazioni di I/O verso le memorie di massa



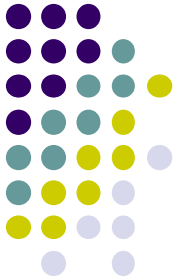
...costituiscono una generalizzazione di quelle primarie

La libreria I/O del C++ mette a disposizione tre classi:

- **ifstream** per l'input;
- **ofstream** per l'output;
- **fstream** per operazioni di aggiornamento;

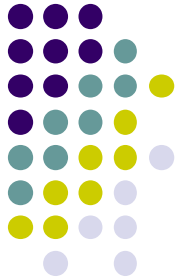
Le classi **ifstream** e **ofstream** sono derivate da **istream** e **ostream** e aggiungono alle operazioni da esse definite anche le operazioni su file ad accesso diretto.

Stream standard di I/O



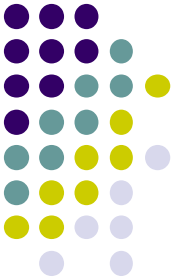
- In C++ un programma comunica con l'esterno mediante i seguenti stream standard:
  - **stdin** standard input (di default associato alla tastiera)
  - **stdout** standard output (di default associato al video)
  - **stderr** standard output per i messaggi (associato al video)
- che rispettivamente rappresentano il canale di immissione dati (tastiera), quello di uscita dei risultati (video) e quello in cui vengono riportati eventuali situazioni di errore causate dall'esecuzione di un comando (di default il video)
- I canali standard di I/O sono "collegati" di default rispettivamente alle variabili **globali di I/O cin, cout e cerr**
- Questi oggetti vengono messi automaticamente a disposizione all'avvio dell'esecuzione di un programma, il sistema provvede automaticamente all'apertura ed alla chiusura dello stream ad essi associato

Operatori di flusso



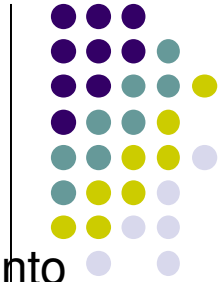
- Utilizzando gli oggetti cin, cout e cerr è possibile effettuare le operazioni di I/O per mezzo degli **operatori di flusso**
  - **<< (inserimento)** i cui operandi di sinistra sono rispettivamente cout oppure cerr (che ha proprietà identiche a quelle di cout); l'operatore << opera una conversione in modo da inserire nello stream di output una sequenza di caratteri
  - **>> (estrazione)** il cui operando di sinistra è cin; l'operatore >> opera una conversione perchè estrae dal flusso di input una sequenza di caratteri e la converte nel tipo dell'operando di destra

File



- Per memorizzare un dato su un'unità di memoria di massa viene utilizzata una variabile di tipo stream chiamata *file*
- Come già detto uno stream viene connesso al dispositivo tramite una operazione di apertura, la connessione viene interrotta chiudendo lo stream

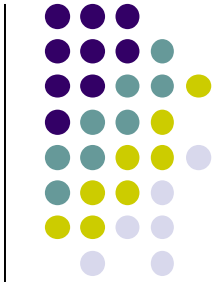
File



- I file posso essere di due tipi:
  - FILE DI TESTO: è una sequenza di caratteri, che durante il trasferimento può subire anche delle conversioni a seconda delle necessità dell'ambiente di destinazione.
  - FILE BINARI: è una sequenza di byte

- **Per leggere o scrivere su un file è necessario:**
  1. Includere nel programma C++ la libreria <fstream>
  2. Dichiarare una variabile di tipo file (fstream); Effettuare il collegamento tra la variabile dichiarata di tipo file e lo specifico flusso da cui vengono acquisite/scritte le informazioni, tale operazione prende il nome di apertura del file e viene effettuata attivando la funzione open. Durante questa operazione è necessario specificare la forma di accesso che si vuole utilizzare: lettura, scrittura o entrambe.
  3. Al termine delle operazioni sul file, lo stream dovrà essere chiuso; questa operazione implica la distruzione del nome del file stesso.

Apertura di un file



Richiede un collegamento tra l'unità fisica in oggetto dell'operazione e la variabile di tipo **stream** definita nel programma.

Due distinti formalismi (coincidenti nella semantica):

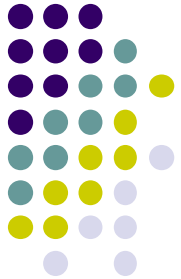
- Attraverso un costruttore

  <tipo file> nomeInterno (nomeEsterno [,<specificatori>]);

- Attraverso la funzione **open**

  <tipo file> nomeInterno;
  nomeInterno.open (nomeEsterno [,<specificatori>]);

Chiusura di un file



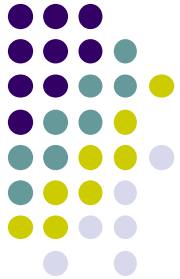
E' complementare a quella di apertura e pone termine al collegamento tra la variabile di tipo file definita nel programma e il file disponibile sulle memorie di massa.

Attraverso la funzione **close;**

<tipo file> nomeInterno;

nomeInterno.close( );

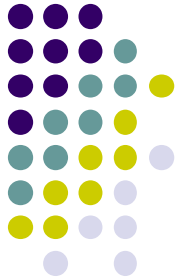
Specifica delle operazioni da compiere



- *Informazioni relative al tipo di operazioni sul file*

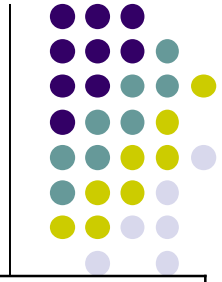
`ios::in`	*Input*
`ios::out`	*Output*
`ios::in | ios::out`	*Input & Output*
`ios::app`	*Output con append*
`ios::ate`	*Dopo* **open** *si sposta a fine file*

Posizionamento in lettura e scrittura



- *fstream* gestisce due puntatori, uno per la lettura e uno per la scrittura che possono essere ri-posizionati usando le apposite funzioni membro:
- posizionamento e interrogazione del puntatore in lettura (g come get):
  - funzione *seekg*
  - funzione *tellg*
- posizionamento e interrogazione del puntatore in scrittura (p come put):
  - funzione *seekp*
  - funzione *tellp*

Funzioni per il posizionamento

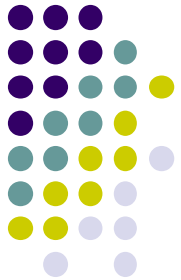


`istream& seekg(long p)`
`istream& seekg(off_type off,`
`ios_base::seekdir way)`
	Posizionamento per lettura

`ostream& seekp(long p)`
`ostream& seekp(off_type off,`
`ios_base::seekdir way)`
	Posizionamento per scrittura

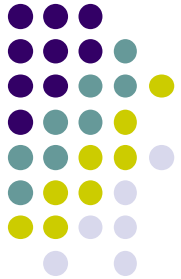
`long tellg()`	*Fornisce posizione per lettura*
`long tellp()`	*Fornisce posizione per scrittura*

Operazioni di lettura e scrittura



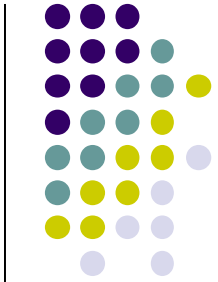
- la posizione in lettura o scrittura si ottiene specificando un indirizzo “assoluto” nel file oppure calcolando l’offset (positivo o negativo) dalla posizione corrente misurato in byte
- La posizione corrente può essere specificata fornendo un secondo argomento alla funzione seekg (seekp) in uno dei seguenti modi:
  - ios_base::beg (spostamento relativo rispetto all'inizio del file)
  - ios_base::cur (spostamento relativo relativo rispetto alla posizione corrente)
  - ios_base::end (spostamentorelativo rispetto alla fine del file)

File di Testo



- L’apertura di un file avviene per default in modalità testo
- Su un file di tipo testo è possibile scrivere e leggere stream di caratteri utilizzando i consueti operatori (<< e >>)

Esempio: scrittura su file testo

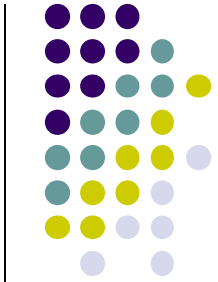


```
#include <iostream>
#include <stdlib.h>
#include <fstream>


using namespace std;

int main(int argc, char *argv[]) {
int vett[10]={0,1,2,3,4,5,6,7,8,9 };
ofstream OutFile;                    // definizione variabile di tipo ofstream
OutFile.open ("./dati.txt");       // apertura
if (!OutFile)                        // controllo
cout << "Errore di apertura del file"<< endl;
else {
for(int i=0;i<10;i++)
OutFile << vett[i]<<endl;           // scrittura
OutFile.close();                     //chiusura
}
system("PAUSE");
  return 0;
}
```

Esempio: lettura da file testo



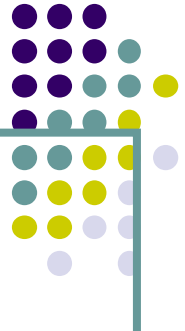
```
#include <iostream>
#include <stdlib.h>
#include <fstream>

using namespace std;

int main(int argc, char *argv[]) {

int vett[10];
ifstream InFile;                    // definizione variabile di tipo ifstream
InFile.open ("./dati.txt");         // apertura
if (!InFile)                    // controllo
cout << "Errore di apertura del file" << endl;
else {
for(int i=0;i<10;i++) {
InFile >> vett[i];                  // lettura
cout << vett[i] << endl;
}
InFile.close();                         // chiusura
}
system("PAUSE");
return 0;
}
```

Esempio: scrittura e lettura da file testo



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

struct Record {
      int codice;
      string nome;
      int numTel;
};
```

```
int main() {
      char a[20];
      Record r;

      cout << "Inserisci il nome del file da creare: ";
      cin >> a;
      ofstream outfile(a, ios::out);

      cout << "inserisci codice, nome e numero di telefono: \n";
      cout << "->";
      while(cin >> r.codice >> r.nome >> r.numTel){
        outfile << r.codice << " " << r.nome << " " << r.numTel << endl;
        cout << "->";
      }
      outfile.close();
```

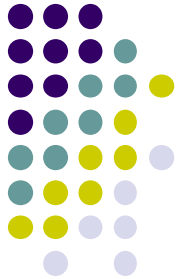
```
cout << "Visualizzazione del file creato: \n";
      ifstream infile(a,ios::in);
      while(infile >>  r.codice >> r.nome >> r.numTel)
          cout << "\n" << r.codice << " " << r.nome << " " << r.numTel;
      cout << endl;
      infile.close();

      system("PAUSE");
      return 0;
}
```

1 Marianna 7682324
2 Alberto 7684568
3 Monica 7623412

file rubrica.txt

Esempio: copia di file (testo) carattere per carattere, v.1



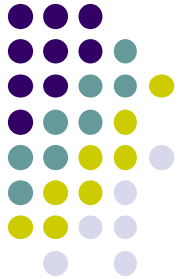
```
#include <iostream>
#include <fstream>
#include <stdlib.h>

#define  DESTINAZIONE  "C:\\CORSI\\Programmazione1\\destinazione.txt"
#define SORGENTE "C:\\CORSI\\Programmazione1\\sorgente1.txt“

using namespace std;

int main(int argc, char *argv[]) {
  fstream file1, file2;
  char c;
  file1.open(SORGENTE, ios::in);
  file2.open(DESTINAZIONE, ios::out);
  while (( c=file1.get())!=EOF)
        file2.put(c);
  file1.close();
  file2.close();
system("PAUSE");
 return 0;
}
```

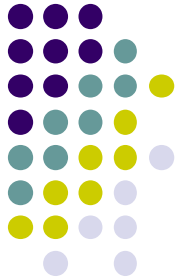
Esempio: copia di file, v.2



```
//…. (include e define come nella versione 1)


int main(int argc, char *argv[]) {
  fstream file1, file2;
  char c;
  file1.open(SORGENTE, ios::in);
  file2.open(DESTINAZIONE, ios::out);
  if (file1&&file2) {
     while (( c=file1.get())!=EOF)
       file2.put(c);
     file1.close();
     file2.close();
   }
   else cout << "\n errore in fase di apertura dei file!";
system("PAUSE");
 return 0;  }
```

Esempio: copia di file, v.3

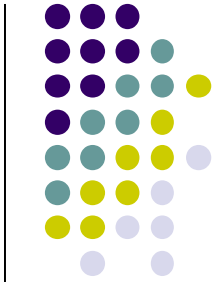


```
//…. (include e define come nella versione 1)


int main(int argc, char *argv[]) {
  fstream file1, file2;
  char c;
  file1.open(SORGENTE, ios::in);
  file2.open(DESTINAZIONE, ios::out|ios::app);

  if (file1&&file2) {
     while (( c=file1.get())!=EOF)
       file2.put(c);
     file1.close();
     file2.close();
   }
   else cout << "\n errore in fase di apertura dei file!";
system("PAUSE");
 return 0;  }
```

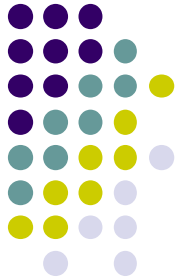
Esempio: copia di file, v.4



```
//…. (include e define come nella versione 1)

int main(int argc, char *argv[])
{
 fstream file1, file2;
 char c;
 char * sorgente=argv[1];
 if(argc != 3) {
  cout <<  “Specificare il nome del file sorgente e del file destinazione" << endl;
  cout <<  "Ripetere l'operazione" << endl;
   }
 else {
 file1.open(argv[1], ios::in);
 if (file1) {
    file2.open(argv[2], ios::out);
    while (( c=file1.get())!=EOF)
       file2.put(c);
    file1.close();
    file2.close();
 }
 else cout << "\n errore: non posso aprire il file sorgente " << sorgente;
 }
 system("PAUSE");
 return 0; }
```

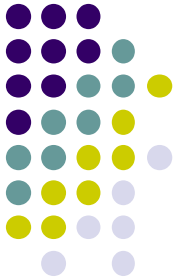
Argomenti della riga di comando in C/C++



- È anche fornire argomenti della riga di comando in C/C++.
  - Gli argomenti della riga di comando sono i valori forniti dopo il nome del programma nella shell della riga di comando dei sistemi operativi.
  - Gli argomenti della riga di comando sono gestiti dalla funzione main() di un programma C/C++.

- Per passare argomenti della riga di comando, in genere si definisce un main() con due argomenti:
  - il primo argomento è il numero di argomenti della riga di comando
  - il secondo è un elenco di argomenti della riga di comando.

Argomenti della riga di comando in C/C++



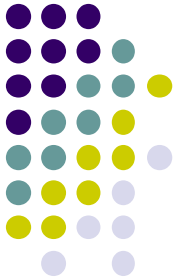
```
int main(int argc, char *argv[]) { /* ... */ }
```

oppure

```
int main(int argc, char **argv) { /* ... */ }
```

- `argc (ARGument Count)` è una variabile intera che memorizza il numero di argomenti della riga di comando passati dall'utente incluso il nome del programma.
  - Es: se passiamo un valore a un programma, il valore di argc vale 2 (uno per l'argomento e uno per il nome del programma)
  - Il valore di argc dovrebbe essere non negativo.
- `argv (ARGument Vector)` è un array di puntatori di caratteri che elenca tutti gli argomenti.
  - Se argc è maggiore di zero, gli elementi dell'array da argv[0] ad argv[argc-1] conterranno puntatori a stringhe.
  - argv[0] è il nome del programma, dopodiché fino ad argv[argc-1] ogni elemento è un argomento della riga di comando.

Riferimenti



- Da "Che C Serve": Capitolo 7, da §7.5 a §7.8