Laboratorio II – 1° Modulo

Lezione 8

Lettura e scrittura di file di dati

Input/Output in C++

- Sin dal primo codice in C++ abbiamo incluso la libreria standard `iostream` che ci ha permesso di acquisire o scrivere sequenzialmente messaggi o dati attraverso il terminale

- Queste informazioni vengono fornite da riga di comando dall'utente e si perdono non appena il terminale o l'eseguibile viene chiuso

- Volendo conservare i risultati prodotti per un uso successivo, risulta necessario scrivere fisicamente (salvare) queste informazioni volatili su di un file (file di caratteri, file binario, immagine, etc.)

- I comandi di I/O su file sono analoghi a quelli su terminale. Dobbiamo includere la libreria `fstream` che contiene le classi di I/O su file: www.cplusplus.com/reference/fstream/fstream

Lettura e scrittura da file

```
#include <iostream>
#include <fstream>
#include <string>

int main () {

        const char *filename = "File.txt";

        // costruisco un oggetto di tipo ofstream passando come parametro di input
        // il nome del file su cui verrà scritto l'output
        std::ofstream OutFile (filename);

        int a = 12;
        double b = 24.5;
        char parola [] = "ciao";

        OutFile << a << "    "<< b << std::endl; // scrivo in output i valori delle variabili a e b
        OutFile << parola << std::endl;
        OutFile.close(); // chiudo il file


        int x=0;
        double y=0;
        std::string word;

        std::ifstream InFile (filename);  // apro il file "File.txt" in modalità lettura

        InFile >> x >> y; // leggo i numeri contenuti nel file e li memorizzo nelle variabili x e y
        InFile >> word;
        InFile.close();

        std::cout << "Il file " << filename <<
                " contiene i numeri: "<< x <<" e " << y << std::endl;
        std::cout << "e la parola: "<< word << std::endl;

        return 0;
}
```

Puntatore a char

Le classi di I/O su file di `fstream`:

- `ifstream` per **input**
- `ofstream` per **output**
- `fstream` per **input/output**

Dichiaro variabile di tipo file ed "apro" il file di nome `filename`

Uso l'operatore `<<` per scrivere l'output sul file già aperto (come si faceva su terminale con `cout`)

Uso l'operatore `>>` per leggere l'output sul file già aperto (come si faceva su terminale con `cin`)

Il file può essere aperto e chiuso più volte in uno stesso programma. Al termine del programma viene chiuso automaticamente

Lettura e scrittura da file

- **Remark 1:**
  I costruttori `ifstream` / `ofstream` / `fstream` prendono in input un puntatore a char:

```
int main (int argc, char** argv) {
        if (argc < 2) {  // verifico che venga passato almeno un parametro da riga di comando
                std::cout << "Digitare il nome del file da riga di comando!\n";
                std::cout << "\t./esempio02 File.txt\n";
                return 1;
        }
        std::string NomeFile = argv[1]; // uso un oggetto string per memorizzare il nome del file
        std::ofstream OutFile (NomeFile.c_str());

        std::ifstream InFile (argv[1]);
...
}
```

Uso il metodo `c_str()` per convertire l'oggetto `string` in un puntatore a `char`

Posso anche usare direttamente `argv[1]` come argomento visto che è un puntatore a `char`

Lettura e scrittura da file

- **Remark 2:**
  Posso usare il metodo `open("NomeFile.txt",OpenMode)` di `fstream` per specificare il tipo di operazione da fare sul file attraverso `OpenMode`

```
#include <iostream>
#include <fstream>

int main ()
  {
    std::fstream OutFile;
    OutFile.open("file.txt",std::ios::out);
    OutFile << "12 24" << std::endl;
    OutFile.close();

    int a,b;
    std::fstream InFile;
    InFile.open("file.txt",std::ios::in);
    InFile >> a >> b;
    InFile.close();

    std::cout << "Ho letto: " << a << "\t" << b << std::endl;
    return 0;
}
```

File aperto in sola scrittura
In `ofstream` l'OpenMode è `std::ios::out` per default

`std::ios::app` si usa per scrivere su un file in modalità *append* (cioè aggiungo righe a quelle già esistenti)

File aperto in sola lettura
In `ifstream` l'OpenMode è `std::ios::in` per default

Se un file ha molte righe...

```
#include <iostream>
#include <fstream>
#include <string>

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

        if (argc < 2) {  // verifico che venga passato almeno un parametro da riga di comando
                std::cout << "Digitare il nome del file da riga di comando!\n";
                std::cout << "\t./esempio03 DataFile.txt\n";
                return 1;
        }

        std::string NomeFile = argv[1];
        std::ofstream OutFile (NomeFile.c_str());

        for (int i=0; i<10; i++) {
                OutFile << i*2 << "\t" << i*2+1<<std::endl;
        }
        OutFile.close();
```

Scrittura di più righe:
`std::endl` interrompe la riga

Se un file ha molte righe...

```
int x=0;
int y=0;
int Nrighe=0;

std::ifstream InFile (NomeFile.c_str());
while (true) {  // imposto un loop infinito, che verrà interrotto con l'istruzione break
        InFile >> x >> y;
        if (InFile.eof()==true) // il metodo eof() restituisce true alla fine del file
                break;

        std::cout << x <<"\t" << y << std::endl;
        Nrighe++;
}
InFile.close();

std::cout << "Il file " << NomeFile <<
        " contiene "<< Nrighe <<" righe"<< std::endl;

return 0;
}
```

Esempio di contenuto del file

1	18
2	17
3	19
4	13
5	12
...	...

Se un file ha molte righe...

Questo codice non funziona (stampa 2 volte l'ultima riga del file):

```
...
while (InFile.eof() == false)
  {
    InFile >> x >> y;
    std::cout << x << "\t" << y << std::endl;
    Nrighe++;
  }
...
```

Il motivo è che per far diventare `InFile.eof() == true` è necessario effettuare l'operazione di lettura anche dopo aver letto l'ultima riga del file. E` un po' come se il file avesse N+1 righe (dove N è il numero di coppie x,y salvate nel file), cioè l'ultima vera riga del file, la N+1-esima, contiene il carattere di "`EOF`" (i.e. `End Of File`)
Dopo ogni riga del file letta, e prima di manipolare i dati (e.g. stamparli a schermo), è necessario quindi effettuare il controllo di `eof() == true`

Tips & tricks

È buona norma, nella scrittura di un codice, inserire dei comandi che verifichino l'integrità (o esistenza) del file aperto:

```
std::ifstream InFile (filename.c_str());

// metodo good() per verificare che sia possibile leggere il file
// se il file non è presente nella cartella o non è possibile aprirlo
// stampo un messaggio di errore
if (InFile.good() == 0)
{
  std::cout << "Errore! Non è possibile aprire il file "<< filename <<std::endl;
  return 1;
}
```

- Esempio di lettura di dati e salvataggio in `std::vector`

```
std::vector<int> vec1; // definisco due vector per memorizzare i numeri contenuti nel file
std::vector<int> vec2; // N.B.: non mi serve sapere a priori quanti numeri dovrò salvare

while (true) { // imposto un loop infinito, che verrà interrotto con l'istruzione break
  InFile >> x >> y;
  if (InFile.eof()==true) // il metodo eof() restituisce true alla fine del file
    break;

  vec1.push_back(x);
  vec2.push_back(y);
  Nrighe++;
}
```

N.B. Non serve conoscere a priori il numero dei dati contenuti nel file

Esercizi

- **Esercizio 1:** Scrivere un codice che legga i dati contenuti nel file 'dati.txt' ed inserirli in due `vector<double>`. Al termine della lettura del file di input, impostare un ciclo `for` per rileggere i numeri salvati e calcolare la media e la deviazione standard dei dati contenuti nei due `vector`. Stampare i risultati in un file di output. Usare `argc` e `argv` per inserire da terminale i nomi dei file di input/output

- **Esercizio 2:** Creare la classe dei vettori geometrici (2D) nel piano. Implementare le operazioni con i vettori (somma, sottrazione, prodotto, etc...) secondo il file di intestazione fornito (`vett2d.h`)

Approfondimenti

- `namespace`
- `std::stringstream`

namespace

I **namespace** permettono di raggruppare oggetti con un nome:

```
namespace spazio
 {
     int a;
     double b;
 }
```

- Per utilizzarli: `spazio::a`
  `spazio::b`

- Proprio come il namespace "`std`": `std::cout << ciao << std::endl;`
- L'uso di `namespace` permette di evitare conflitti di nomi di identificatori (variabili, funzioni, strutture, etc.), soprattutto in codici grandi e complessi, scrivendo così codici più sicuri e di più facile *debugging*
- Si può comunque omettere il riferimento al `namespace` attaverso l'istruzione `using namespace`, da riportare prima dell'inizio del main, ad esempio: **`using namespace std;`**

std::stringstream

- Gli oggetti di tipo `stringstream` hanno a disposizione dei metodi molto utili per manipolare flussi di dati, potendo gestire sia numeri che parole in maniera analoga ai metodi `std::cin` e `std::cout`
- Includiamo `sstream`

```
#include <iostream>     // std::cout
#include <sstream> // std::stringstream

using namespace std;

int main () {

  stringstream ss;
  int anno = 1923;
  string nome = "Jonh";

  // posso unire numeri e stringhe in un unico stream
  ss << nome << " nacque nel " << anno;

  // conversione sstream -> string
  string frase = ss.str();
  cout << frase << endl;

  return 0;
}
```

N.B.: per inizializzare una `stringstream` utilizzare la coppia di metodi:

```
ss.clear();
ss.str("");
```

Con `<<` scrivo sulla variabile `stringstream` (come con `std::cout`)

Lettura file con `stringstream`

```
int main(int argc, char** argv)
{
        string filename = argv[1];

        ifstream in (filename.c_str());
        // metodo good() per verificare che sia possibile leggere il file
        if (in.good()==0)  {
                cout << "Errore! Non è possibile aprire il file "<< filename <<endl;
                return 1;
        }

        // uso il metodo getline (istream& is, string& str)
        // per leggere il file una riga alla volta
        //(restituisce falso alla fine del file)

        string line, parola;
        stringstream ss;
        int nRighe=0;
        while (getline(in, line))
        {
                nRighe++;
                cout << "\nRiga " << nRighe << ": "<< line << endl;
                cout << "Selezione parole pari: ";
                int nParole=0;
                ss << line; // riempio lo stringstream con la riga
                while (ss >> parola) // leggo parola per parola
                {
                        nParole ++ ;
                        if ((nParole%2) == 0)
                                cout << parola << " ";
                        else continue;
                }
                cout << endl;
                ss.clear(); // svuoto lo stringstream
        }

        return 0;
}
```

Una variabile `stringstream` può essere usata per I/O da file

Metodo `getline` prende in input un oggetto `ifstream` (file da leggere) e un oggetto `string`, in cui memorizzare le righe del file
Restituisce **`false`** alla fine del file

Con `>>` leggo lo stream come farei con `std::cin`

In questo esempio abbiamo deciso di selezionare solo le parole che occupano una posizione pari all'interno della riga