Gestione File in C++

4BIA – AS 2021-22

Riferimento:

https://www.cplusplus.com/reference/fstream/fstream/

`#include <fstream>`	Per lavorare con i file abbiamo bisogno di includere la libreria **fstream**
`fstream f;`	Dichiarazione di un oggetto di tipo fstream: **f** contiene tutte le informazioni ed i metodi per lavorare con un file
`f.open("nomefile", mode);`	Il metodo **open()** della classe fstream ci consente di aprire un file su disco. Il primo parametro rappresenta il nome del file da aprire, ad esempio: **".\\nomefile.txt"** Il secondo parametro (mode) è la modalità di apertura del file, che può essere: • **ios::in**, apertura in input (lettura di file) • **ios::out**, apertura in output (scrittura di file) • **ios::app**, apertura in append (continua a scrivere in coda al file)

`f.eof();`	E’ il metodo che ritorna **true** se si è arrivati alla fine del file, **false** altrimenti
`f.is_open();`	E’ il metodo che ritorna **true** se l’apertura del file è andata a buon fine, **false** se si sono verificati degli errori nell’operazione di apertura del file
`f.close();`	Dopo aver completato le operazioni di I/O (input/output) su file, dobbiamo invocare il metodo **close()** per rilasciare le risorse associate al file

`int numero;`
`f>>numero;`
 	
`string parola;`
`f>>parola;`
	Utilizziamo **f** come una sorgente di input per estrarre il contenuto da un file di testo.
	
	In questi casi si presuppone che il file contenga una sequenza di numeri interi o di parole

`string word;` `getline(f, word, ',');`	La funzione **getline()** consente di leggere dal file **f** e inserire in una stringa **word** le sequenze di caratteri nel file separati da un carattere **delimitatore** che può essere: • lo spazio, `' '` • una virgola, `','` • il ritorno a capo, `'\n'` • punto e virgola, `';'` • `eccetera` Questa funzione è utile per gestire i file CSV, in cui i vari campi di ogni riga sono separati da ',' oppure da ';' (tranne l'ultimo campo che termina con un ritorno a capo '\n').

`int numero;`
 	
`f<<numero<<" ";`
	Utilizziamo **f** come la destinazione dell'output e inviamo ad f i dati da memorizzare nel file, che possono essere numeri, parole, eccetera.

`typedef struct p{`
`string nome;`
`int altezza;`
`} Persona;`
 	
`Persona friend;`
`f<<friend.nome<<",”<<friend.altezza<<endl;`
	Se si deve salvare in un file di testo di tipo CSV il contenuto di una tabella, ovvero di un array di struct, ogni campo della struct sarà salvato utilizzando il carattere `',' ` come delimitatore

`fstream f;` `f.open("dati.dat", ios::out|ios::binary);`	Modalità di apertura di un file **binario** in output (scrittura);

`tipoDato varName;`
 	
`f.write((char *)&varName, sizeof(tipoDato));`
	Per salvare dati in un file binario dobbiamo usare il metodo **write**:
	- il primo parametro è il puntatore alla variabile da salvare (con un cast a **char *** dell'indirizzo della variabile)
	- il secondo parametro è la dimensione del dato, espresso in numero di byte
	
	Il metodo **write** inizia a leggere dalla posizione indicata da **(char *)&varName** e legge i successivi **sizeof(tipoDato)** byte memorizzandoli nel file

`tipoDato array[N];` `f.write((char *)array, sizeof(tipoDato)*N);`	Se dobbiamo salvare un array di elementi omogenei, specifichiamo la dimensione dell'array **(sizeof(tipoDato)*N)** a partire dall'indirizzo base **(array)**

`fstream f;` `f.open("dati.dat", ios::in|ios::binary);`	Modalità di apertura di un file **binario** in input (lettura);

`tipoDato varName;`
 	
`f.read((char *)&varName, sizeof(tipoDato));`
	Per leggere dati da un file binario dobbiamo usare il metodo **read**:
	- il primo parametro è il puntatore alla variabile in cui memorizzare i dati letti dal file (con un cast a **char ***)
	- il secondo parametro è la dimensione del dato letto, espresso in numero di byte
	
	Il metodo **read** inizia a leggere dalla posizione corrente nel file e salva nella variabile indicata il contenuto letto

`tipoDato array[N];` `f.read((char *)array,sizeof(tipoDato)*N);`	se la lettura riguarda una sequenza di dati omogenei che memorizziamo in un array

`f.seekg(0, f.end);`	In un file aperto in lettura, posizioniamo la testina di lettura alla fine del file. **seekg()** è il metodo che posiziona la testina di lettura al byte **0** rispetto alla fine del file **f.end**
`int nbytes = f.tellg();`	Il metodo **tellg()** ritorna il numero di bytes dalla posizione iniziale del file fino alla posizione corrente della testina di lettura. Se siamo alla fine del file, tellg() ritorna il numero di byte nel file binario.
`int nElementi = nbytes/sizeof(tipoDato);`	Poiché sappiamo il tipo dei dati memorizzati nel file binario, dividendo il numero totale di byte nel file per la dimensione di ogni singolo dato otteniamo il numero di elementi memorizzati nel file
`f.seekg(0, f.beg);`	Riposiziona la testina di lettura al primo byte rispetto all'inizio del file
`int i = 1;` `f.seekg(i*sizeof(tipoDato), f.beg);`	Se **i** è l'indice dell'element che vogliamo leggere dal file binario, dobbiamo posizionarci al byte corrispondente rispetto all'inizio del file che si calcola come **(i*sizeof(tipoDato))**