Programmazione a oggetti

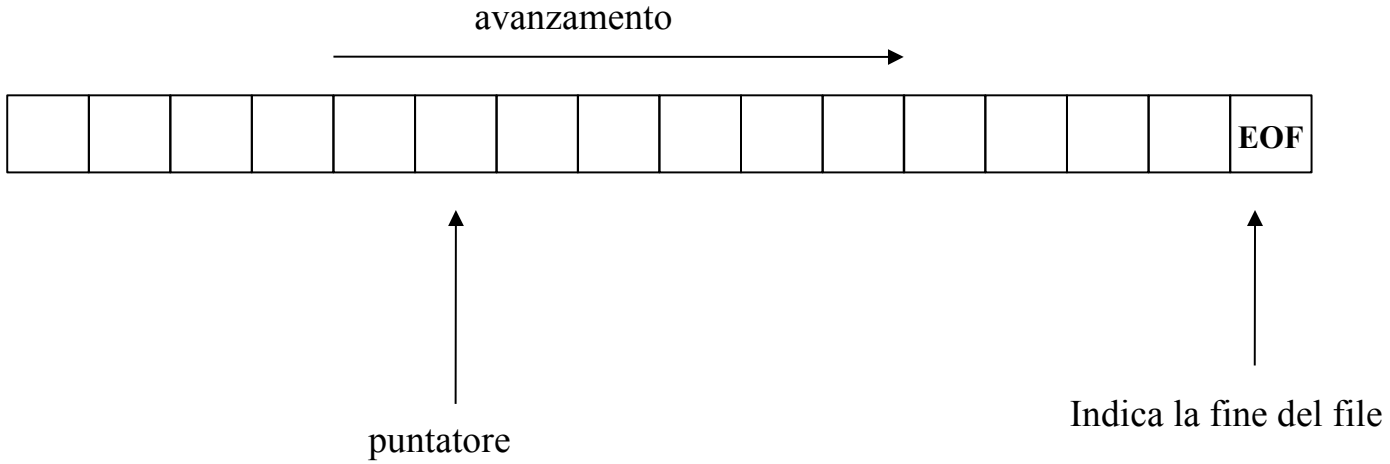
I/O del C++

Bibliografia
Aguilar, Capitolo 15

A.A. 2020/2021
Francesco Fontanella

Gli stream di I/O del C++

- Uno stream è un interfaccia logico che è indipendente dal particolare dispositivo di input o output.
- Esistono due tipi di stream:
  - di testo: costituiti da una sequenza di caratteri ASCII;
  - binari: possono essere utilizzati per qualsiasi tipo di dato
- Accesso sequenziale:



Stream Standard

- In C++ un programma comunica con l'esterno mediante i seguenti stream standard:
  - **stdin** standard input (default: tastiera)
  - **stdout** standard output (default: video)
  - **stderr:** standard output per i messaggi di errore, associato di default al video, ma può essere associato, ad esempio, ad un file di log

- I canali standard di I/O sono “collegati” di default rispettivamente alle variabili globali di I/O: **cin, cout e cerr**.

Operatori di Flusso

- In C++ l'accesso agli stream è effettuato per mezzo dei cosiddetti **operatori di flusso**.

- Questi operatori sono delle funzioni che hanno una particolare sintassi.

- L'aspetto importante di questi operatori è che sono sovraccaricati (overloaded):
  - Lo stesso operatore può essere utilizzato per qualunque tipo di dato.

Le Variabili Stream

- **Input**: per aprire un file di input è necessario dichiarare una variabile di tipo ifstream:

```
ifstream in;
```

- **output**: per aprire un file di output è necessario dichiarare una variabile di tipo ofstream:

```
ofstream out;
```

- **Input e output**; per aprire un file sia in input che output è necessario dichiarare una variabile fstream

```
fstream io;
```

Specificatori di accesso

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

Apertura e chiusura di File

- Uno stream viene connesso ad un dispositivo di I/O tramite una operazione di apertura:

```
<tipo stream> stream_name;

stream_name.open(nomefile[,<specificatori>]);
```

- La connessione viene interrotta con una operazione di chiusura:

```
<tipo stream> stream_name;

stream_name.close();
```

Le funzioni get e put

- Per accedere ai caratteri di un file si può usare la funzione get. Ne esistono due versioni:

```
fstream& stream_name.get(char c)
```

```
char stream_name.get()
```

`stream_name` è lo stream dal quale si vogliono estrarre i caratteri,

`c` è la variabile in cui si vogliono memorizzare i caratteri estratti

- La funzione put() inserisce dei caratteri in uno stream:

```
fstream stream_name.put(char c)
```

Copiare un file

```
int copy_file(ifstream &in, ofstream &out)
{
  char c;
  int i;

  i=0;
  while (( c=in.get())!= EOF) {
    out.put(c);
    i++;
  }
  return i;
}
```

Passaggio per riferimento

NOTA
Gli stream vanno passati sempre per riferimento: se la funzione a cui è passato lo modifica, la modifica deve essere vista anche dal chiamante della funzione.


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

const char source_file[] = "C:\CORSI\PO14\sorgente.txt“;
const char dest_file[]   = "C:\CORSI\PO14\destinazione.txt";

using namespace std;

int main() {
  fstream instream, outstream;

  instream.open(source_file, ios::in);
  outstream.open(dest_file, ios::out);

  if (!instream) {
    cout<<endl<<”impossibile aprire il file: ”<<source_file;

    return;
  }
```


```
  if (!outstream) {
    cout<<endl<<"Impossibile aprire il file: "<<dest_file;

    return;
  }

  // Si chiama la funzione di copia
  copy_file(instream, outstream);

  instream.close();
  outstream.close();

 return 0;
}
```

Da File a vettore

```
void file2vec(ifstream &in, TipoValue v[], int &n)
{
  n=0;
  while(in>>v[n])
   n++;

  return;
}
```

Da vettore a file


```
int vec2file(fstream &out, TipoValue v[], int n)
{
  int i;

  i=0;
  while (i < n && out)
    out<<v[i++]<<" ";


  return i;
}
```

Differenza tra >> e get

- Qual' è la differenza tra questi due programmi?:

```
int main() {
  char c;

  while(true) {
    cin >> c;
    if (cin.eof()) //Ctrl-D da
tastiera
      break;
      cout << c;
    }
    cout<<endl;

  return 0;
}
```

```
int main() {
   char c;

   while(true) {
     c = cin.get();
     if (c == EOF)
       break;
     else cout << c;
   }

   cout<<endl;

   return 0;
 }
```

La Funzione getline


```
ifstream& getline(char *str, int num);
```

- Legge un'intera riga dallo stream e la copia in `str`
- Aggiunge in maniera automatica il carattere nullo ('\0').
- Il carattere di fine riga viene estratto dallo stream ma non inserito in `str`
- Se non si specifica nessuno stream la chiamata fa riferimento alla stdin


```
ifstream& getline(char *str, int num, char delim)
```

- Quest'altra forma si ferma per le stesse condizioni della precedente, più un ulteriore condizione:
  - La lettura dallo stream del carattere `delim`

DOMANDA

qual'è il codice della funzione getline?

Getline: esempio

- Dato un un file di testo, si vuole scrivere un programma che visualizzi solo le linee del file che contengono una determinata stringa. Visualizzando inoltre il numero della linea

- A tal fine sarà utile la funzione strstr:

```
const char *strstr (const char *str1,  const
                    char *str2);
```

  che ritorna un puntatore alla prima occorrenza di `str2` in `str1`


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

const int max_name = 256;
const int max_line = 1000;

int main ()
{
   fstream instream;
   char file_name[MAX_NAME], line[max_line], str[max_line];
   bool found;

   cout<<endl<<"digitare il nome del file:"<<endl;
   cin.getline(file_name, max_name);
```

SEGUE...


```
instream.open(file_name); // Si apre il file in lettura
if (!instream) {
  cout<<endl<<"impossibile aprire il file: "<<file_name;
  exit(EXIT_FAILURE);
}

cout<<endl<<"inserisci la stringa da cercare:"<<endl;
cin.getline(str, max_line);

i=1
found = false;
while (instream) {
   instream.getline(line, max_line);
   if (strstr(line, str) != 0) {
     cout<<i<<" "<<line<<endl;
     found = true;
   }
   ++i;
}
```

SEGUE...


```
  if (!found)
    cout<<endl<<"la stringa: "<<str<< "è   assente"<<endl;

  instream.close();

  return 0;
}
```

Getline: acquisire vettori

- La funzione getline può anche essere usata per acquisire un insieme di valori, separati dallo stesso carattere detto **delimitatore**:

```
void line2vec(ifstream &in, int v[], int &n,
              char delim)
{
   char str[max_line];
   n=0;
   while (in.getline(str, max_line, delim))
     v[n++] = atoi(str); //string -> int
}
```

SEGUE ...

```
void line2vec(fstream &in, float v[], int &n,
              char delim)
{
  char str[max_line];

  n=0;
  while (in.getline(str, MAX_LINE, delim))
    v[n++] = atof(str);  // string -> float
}
```

atoi e atof

- Le funzioni atoi e atof:

```
int atoi (const char *str);

float atof (const char *str);
```

  restituiscono il numero intero (float) contenuto all'interno della stringa passata come parametro.

- Eventuali spazi che precedono il numero da convertire vengono scartati.

La funzione flush

- Nelle operazioni di output i dati vengono copiati in un buffer intermedio fino al suo completo riempimento. Solo allora il contenuto viene fisicamente scritto (forse*) sul dispositivo.
- È possibile forzare la scrittura fisica sul dispositivo anche prima del riempimento del buffer, per mezzo della funzione flush.

```
fstream out_stream;
int a;
.
.
out_stream<<a;
out_stream.flush();
```

```
fstream out_stream;
int a;
.
.
out_stream<<a<<flush
```

La funzione ignore

- Consente di eliminare dagli stream di input 1 o anche più caratteri:

  Esempi

Si elimina un solo carattere

```
fstream in_stream;
int a;
   .
   .
in_stream>>a;
in_stream.ignore();
```

Si eliminano 3 caratteri

```
fstream in_stream;
int a;
.
.
in_stream>>a;
in_stream.ignore(3);
```

Controllo dello stream

iostate value (member constants)	indicates	functions to check state flags				
		good()	eof()	fail()	bad()	rdstate()
goodbit	No errors (zero value iostate)	true	false	false	false	goodbit
eofbit	End-of-File reached on input operation	false	true	false	false	eofbit
failbit	Logical error on i/o operation	false	false	true	false	failbit
badbit	Read/writing error on i/o operation	false	false	true	true	badbit

I/O formattato

- La formattazione dell'I/O avviene per mezzo di una serie di flag associati ad ogni stream.
- **Esempi**
  - `Oct`: visualizza in codifica ottale
  - `Hex`: visualizza in codifica esadecimale
  - `Left/right`: allineamento a sinistra/destra
  - `Scientific`: visualizzazione con notazione scientifica

La funzione setf

- I valori dei flag possono essere modificati per mezzo della funzione:

```
fmtflags setf(fmtflags flag)
```

  La funzione restituisce il valore precedente del flag.

- **Esempio**:

```
cout.setf(ios::left)
```

  le cifre successive verranno visualizzate con allineamento a sinistra.

I/O formattato

- In C++ è possibile definire i possibili parametri di formattazione:
  - **width**: ampiezza minima (in termini di #caratteri) del campo da stampare
  - **precision**: numero totale di cifre da visualizzare
  - **fill**: specifica il carattere di riempimento

I/O formattato: esempio

```
#include <iostream>

using namespace std;
int main () {
  double f = 12343.1415912321;

  cout.setf(ios::left);
  cout.fill('*');
  cout.precision(5);
  cout.width(10);
  cout << f << '\n';
  cout.setf(ios::right);
  cout << f << '\n';

  cout.precision(10);
  cout << f << '\n';

  return 0;
}
```

OUTPUT

```
12343*****
*****12343
12343.14159
```

Memorizzazione di struct

```
struct s {
  Tipo1 campo1;
  Tipo2 campo2;
      .
      .
  TipoN campoN
  output(ostream out);
};
```

Il tipo ostream può essere usato per l'output sia su file che su stdout (cout)

Input e output di struct


```
const int max_string = 10;

struct studente {
        char nome[max_string];
        char cognome[max_string];
        int  matr;
        void input();  // input da utente
        bool input(ifstream &in); // input da file
        void output(ostream &out); // output su file/schermo
};
```

Output di struct su file

- Per memorizzare in un file di testo una struct, possiamo immaginare di scrivere una struct per ogni riga e di separare i singoli campi per mezzo di un particolare carattere.

MEMORIA

a	n	t	o	n	i	o	\0		R	o	s	s	i	\0			2148	M	a	r	i	a	\0			N	a	r	d	i	\0			1845	E	l	e	n	a	\0			B	i	a	n	c	h	i	\0	4561

studenti.txt

Antonio, Rossi, 2148
Maria, Nardi, 1845
Elena, Bianchi, 4561

```
void studente::output(ofstream &out)
{
    out<<endl;
    out<<nome<<",";
    out<<cognome<<",";
    out<<matr;

   return;

}
```

Input di struct da file

- Per effettuare l'input della struct studente da file è necessario tenere conto del modo in cui i dati sono stati memorizzati dalla funzione output (ofstream out).
- Poniamo che la struct studente sia stata memorizzata in un file di testo come visto in precedenza.
- Sappiamo inoltre che i campi sono separati dalla virgola
- Per caricare i campi di una singola struct possiamo usare la funzione:

```
getline (char *buf, int num, char delim)
```

La funzione input

```
bool studenti::input(ifstream &in)
{
    char str[max_string];

    if (in.getline(str, max_string, ','))
      strcpy(nome, str);
    else return false;

    if (in.getline(str, max_string, ','))
      strcpy(cognome, str);
    else return false;

    if (in.getline(str, max_string))
      matr = atoi(str);
    else return false;

    return true;
}
```

La matricola NON è seguita dalla virgola.

Restituisce true se è riuscita a leggere dallo stream tutti i campi della struct, false altrimenti

La funzione input: esempio


```
#include <iostream>

#include <fstream>

#include <stdlib.h>

const int max_name = 256;
const int max_students = 1000;

studente std_array[max_students];

int main()
{
  ifstream in;
  char in_name[max_name];
  int n;

  cout<<endl<<"scrivere il nome del file di input: ";
  cin.getline(in_name, max_name);
```


```
  if (!in) {
    cout<<"ERRORE!: impossibile aprire il file: "<<in_name;
    exit(EXIT_FAILURE);
  }

  n=0;
  while (std_array[n].input(in))
    n++;

  cout<<endl<<"sono stati caricati "<<n<<" studenti";
    .
    .
  // Corpo del programma
    .
    .
 return 0;
}
```

I/O Binario

```
int main()
   ifstream in;
   ofstream out;
        .
        .
   in.open(f_name, ios::binary);
        .
        .
   out.open(f_name, ios::binary);
        .
        .
   in.close();
   out.close();
```

- Poniamo di avere un array di struct studente e di volerlo salvare in un file in modalità binaria. A tale scopo possiamo usare la funzione:

```
void write_students(studente s[], int n, ofstream &out)
{
  int dim, i;
  char *ptr;

  dim = n * sizeof(studente);
  ptr = (char *) s;

  for(i=0; i < dim; ++i)
     out.put(ptr[i]);

  return;
}
```

casting per effettuare l'inserimento byte a byte

- Poniamo di avere memorizzato in un file un array di struct studente e di volerlo caricare in un array. A tale scopo possiamo usare la funzione:

```
void read_students(studente s[], int &n, ifstream &in)
{
  int max, i;
  char *ptr;

  max = max_students * sizeof(studente);
  ptr = (char *) s; // CASTING

  i=0;
  while((in.get(ptr[i])) && (i < max))
    ++i;

  n = i / sizeof(studente);

  return
}
```

- Possiamo generalizzare quanto visto prima, ad una generica area di memoria. A tale scopo possiamo usare la funzione:

```
void write_memory(void* ptr, int size, ofstream &out)
{
  int i;
  char *p;


  p = (char *) ptr;

  for(i=0; i < size; ++i)
     out.put(p[i]);

  return;
}
```

casting per effettuare l'inserimento byte a byte

- Possiamo generalizzare quanto visto prima, ad una generica area di memoria. A tale scopo possiamo usare la funzione:

```
int read_memory(void* ptr, int max_size, ofstream &out)
{
  int i;
  char *p;


  p = (char *) ptr;

  while((in.get(ptr[i])) && (i < max_size))
    ++i;

  return;
}
```

casting per effettuare l'inserimento byte a byte

- Le funzioni precedenti diventano:

```
void write_students(studente s[], int n,
   ofstream &out)
{
  int dim, i;
  char *ptr;

  dim = n * sizeof(studente);
  ptr = (char *) s;

  write_memory(ptr, dim, out)

  return;
}
```

```
void read_students(studente s[], int &n,
   ifstream &in)
{
  int max, i;
  char *ptr;

  max = max_students * sizeof(studente);
  ptr = (char *) s; // CASTING

  read_memory(ptr, max, in);

  return
}
```