Lezione 05: Libreria iostream e funzioni di I/O

Introduzione

La libreria standard `iostream` fornisce gli strumenti per gestire input e output **formattati** basati su caratteri. Le operazioni di input sono **tipizzate** e **estensibili**, consentendo così il supporto anche per **tipi definiti dall'utente**.

Tuttavia, altre forme di interazione con l'utente, come le **interfacce grafiche (GUI)**, **non** fanno parte dello standard ISO del C++ e, di conseguenza, **non sono incluse** nella libreria `iostream`.

Output

Un **output stream** (ostream) consente di convertire un oggetto tipizzato in uno **stream di caratteri (byte)**. È possibile definire l'output personalizzato anche per i tipi definiti dall'utente, estendendo il comportamento predefinito.

L'**operatore** `<<` è definito per tutti gli oggetti della classe `ostream`. Di default, i valori scritti su uno stream di uscita vengono convertiti in una sequenza di caratteri.

Ogni operazione con l'operatore `<<` restituisce lo stream stesso, permettendo così la **concatenazione** di più operazioni in un'unica istruzione.

```
int i = 23;
cout << "Risultato " << i << '\n';
```

Input

Un **input stream** (instream) consente di convertire uno stream di caratteri in oggetti di tipo specifico. Nella libreria standard del C++, instream è già implementato per i tipi *built-in*, ma può essere esteso anche per i tipi definiti dall'utente.

L’**operatore** `>>` è definito per tutti gli oggetti della classe `istream`, tra cui anche `cin`. Per leggere una sequenza di caratteri si utilizza il tipo `string`. Tuttavia, quando si usa `>>`, la lettura si interrompe al primo carattere non alfanumerico (come uno spazio o un simbolo).

Per leggere **un’intera riga** di testo, compresi gli spazi, è preferibile usare la funzione `getline()`. Se invece si desidera leggere **un singolo carattere**, si può utilizzare la funzione `get()`.

I/O per i tipi definiti dall’utente

Quando si lavora con i **tipi definiti dall'utente** in C++, è possibile estendere le funzionalità di **input** e **output** per consentire la lettura e la scrittura di oggetti di questi tipi attraverso gli stream (come `cin` e `cout`).

Per fare ciò, si sovraccaricano gli operatori di stream, ovvero l'operatore `>>` per l'input e `<<` per l'output. In questo modo, è possibile definire come un oggetto personalizzato deve essere letto o scritto su uno stream di caratteri.

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

class Punto {
public:
    int x, y;

    Punto(int x = 0, int y = 0) : x(x), y(y) {}

    // Sovraccarico dell'operatore di output (<<)
    friend ostream& operator<<(ostream& os, const Punto& p) {
        os << "(" << p.x << ", " << p.y << ")";
        return os;
    }

    // Sovraccarico dell'operatore di input (>>)
    friend istream& operator>>(istream& is, Punto& p) {
        is >> p.x >> p.y;
```

```
        return is;
    }
};

int main() {
    Punto p1(3, 4);
    cout << "Punto p1: " << p1 << endl;  // Output: (3, 4)

    Punto p2;
    cout << "Inserisci un punto (x y): ";
    cin >> p2;  // Input: 5 6
    cout << "Punto p2: " << p2 << endl;  // Output: (5, 6)

    return 0;
}
```

Gestione degli errori

Gli oggetti di tipo `istream` e `ostream` hanno **flag di stato** che indicano se si è verificato un errore:

Flag	Significato
good()	Nessun errore, tutto ok
eof()	Raggiunta la fine del file (o dell'input)
fail()	Operazione fallita(es. input non valido per il tipo)
bad()	Errore grave sullo stream(es. problemi hardware e I/O)

Forniamo di seguito un esempio dei concetti appena presentati:

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

```
int main() {
    int numero;

    cout << "Inserisci un numero intero: ";
    cin >> numero;

    if (cin.fail()) {
        cout << "Errore: input non valido!" << endl;
        cin.clear(); // Resetta lo stato di errore
        cin.ignore(10000, '\n'); // Scarta i caratteri errati nel buffer
    } else {
        cout << "Hai inserito il numero: " << numero << endl;
    }

    return 0;
}
```

Di seguito elenchiamo le funzioni utili per la gestione degli errori:

- `clear()`: Resetta i flag di errore.
- `ignore(n, delim)`: Scarta fino a `n` caratteri o fino al carattere `delim`.
- `peek()`: Guarda il prossimo carattere senza rimuoverlo dallo stream.
- `getline()`: Utile per evitare problemi con gli spazi in input.