Lezione 04: Tipi definiti dall’utente

Introduzione

Fino ad adesso abbiamo visto solamente tipi **built-in,** che sono di livello piuttosto basso, vicino al livello macchina, un pò scomodi per programmare.
Possiamo quindi attuare dei
meccanismi di astrazione per costruire dei tipi più vicini alle esigenze del programmatore, ossia i cosiddetti **tipi definiti dall’utente (user-defined types).**

Struct

Una **struct (struttura)** è un tipo composito che può contenere variabili (dette membri) e, in C++, anche funzioni, costruttori, ecc...

È molto simile ad una class, ma con una differenza di default nella visibilità.

Differenza	**struct**	**class**
Visibilità predefinita	pubblica (**public**)	privata (**private**)
Uso tipico	Dati semplici, POD	Programmazione OO
Possibilità	Identiche! Può avere: costruttori, metodi, ereditarietà, operatori, ecc...	

Ecco di seguito un esempio di base di questo costrutto:

```
struct Punto {
    int x;
    int y;

    void stampa() {
        std::cout << "(" << x << ", " << y << ")\n";
    }
};
```

```
int main() {
    Punto p1{10, 20};
    p1.stampa();  // stampa: (10, 20)
}
```

Anche le struct possono ereditare da altre classi o struct.

Class

Una classe è un **modello** (template) per creare oggetti. Esso definisce:

- dati (membri) → variabili
- comportamenti (metodi) → funzioni

```
class Persona {
public:
    std::string nome;
    int eta;

    void saluta() {
        std::cout << "Ciao, sono " << nome << " e ho " << eta << " anni\n";
    }
};
```

Elenchiamo di seguito i modificatori di accesso:

Modificatore	**Significato**
public	Accessibile ovunque
private	Accessibile solo internamente
protected	Accessibile da classi derivate

Di **default,** i membri di una classe sono private.

Le classi sono di solito fornite di un **costruttore** che inizializza l'oggetto e di un **distruttore** che viene chiamato quando l'oggetto viene distrutto.

Incapsulamento

Di solito lo specificatore private viene utilizzato per nascondere i dettagli interni, invece lo specificatore public viene utilizzato per esporre ciò che vogliamo mostrare all'esterno.
Questo processo prendere il nome di **incapsulamento**.

```
class ContoBancario {
private:
    double saldo;

public:
    ContoBancario(double iniziale) : saldo(iniziale) {}

    void deposita(double importo) {
        saldo += importo;
    }

    double getSaldo() const {
        return saldo;
    }
};
```

Ereditarietà

Ovviamente nell'ambito delle classi è possibile utilizzare il concetto di ereditarietà per permettere ad una classe di estenderne un'altra.

```
class Animale {
public:
    void dorme() { std::cout << "Zzz...\n"; }
};

class Cane : public Animale {
public:
```

```
    void abbaia() { std::cout << "Bau!\n"; }
};
```

Polimorfismo

Anche il concetto di **polimorfismo** ritorna in C++ e ci permette di usare puntatori a classe base per oggetti di classi derivate. Serve il virtual oppure da C++ 11 l'override esplicito.

```
class Forma {
public:
    virtual void disegna() const {
        std::cout << "Disegna forma generica\n";
    }
};

class Cerchio : public Forma {
public:
    void disegna() const override {
        std::cout << "Disegna cerchio\n";
    }
};

void stampa(Forma* f) {
    f→disegna();  // chiama dinamicamente la funzione giusta
}
```