Prog. Avanzata

Perché non ce n'è di **Roberti** per tutti i *frocioni* (T^T).

`struct` vs `class`

In breve, le strutture e le classi in C++ sono uguali perché C++ smerdo. La principale differenza è che di default gli attributi delle strutture sono `public`, mentre nelle classi `private` (penso).

Costruttori e Distruttori

Per capire che cazzo fanno basta sapere un minimo di italiano. A noi interessa solo come usarli:

```
class Foo {
    public:
        Foo() { std::cout << "Costruttore\n"; }
        ~Foo() { std::cout << "Distruttore\n"; }
};
```

Deep Copy vs Shallow Copy

La **Deep Copy** (in italiano copia profonda) è una copia che copia tutto. A differenza della **Shallow Copy** (copia superficiale), la Deep Copy copia anche gli attributi che sono puntatori (il valore puntato e non l'indirizzo).

Implementazione smerda di una Deep Copy

```
class Foo {
    private:
        int *p;
    public:
        Foo(const Foo &f) {
            p = new int;
            *p = *f.p;
        }
};
```

Passaggio per riferimento in C++

Il passaggio per riferimento in C++ si fa con `&`. Esempio:

```
void foo(int &a) {
    a = 10;
}
```

Differenza tra passaggio per riferimento in C e C++

In C++ il passaggio per riferimento è più figo perché non devi usare i puntatori, ma la sintassi è una merda come per il resto del C++ moderno. Cosa cambia al lato pratico? Se passi una classe o una struttura usi il `.` al posto della `->` per accedere agli attributi.

Non hai capito un cazzo?

Non ti preoccupare, eccoti un array di puntatori a funzioni:

```c
#include <stdio.h>

void foo() { printf("foo\n"); }
void bar() { printf("bar\n"); }

int main(void) {
    void (*f[2])() = { foo, bar };
    return 0;
}
```

P.S. dal vivo avevo spiegato più roba. Chi sta leggendo le slide da github si è perso un sacco di cose.