Prog. Avanzata

"Usa `int` e non `char` che i caratteri in C danno problemi11!!!1!1!"

“

Ho pippato il cazzo con la lista di inizializzazione.

”

~ Mirkolo 🍻

Overloading degli operatori di paragone

In C++ è possibile definire gli operatori di paragone per una classe. Gli operatori di paragone sono operatori binari che confrontano due oggetti e restituiscono un valore booleano.

```cpp
class Foo {
    public:
        int a;
        int b;

        Foo(int a, int b) : a(a), b(b) { }

        bool operator==(const Foo& other) const {
            return a == other.a && b == other.b;
        }

        bool operator!=(const Foo& other) const {
            return !(*this == other);
        }
};
```

Overloading degli operatori di paragone

Nell'esempio precedente, abbiamo definito gli operatori `==` e `!=` per la classe `Foo`. Gli operatori di paragone vengono definiti come metodi della classe e prendono come argomento un riferimento costante all'oggetto da confrontare. Ciò vale anche per `<`, `>`, `<=` e `>=`.

Overloading degli operatori di incremento e decremento

In C++ è possibile definire gli operatori di incremento e decremento per una classe. Gli operatori di incremento e decremento sono operatori unari che incrementano o decrementano il valore di un oggetto.

```
class Foo {
    public:
        int a;

        Foo(int a) : a(a) { }

        Foo& operator++() {
            a++;
            return *this;
        }

        Foo operator++(int) {
            Foo tmp(*this);
            operator++();
            return tmp;
        }
};
```

Overloading degli operatori di incremento e decremento

Nell'esempio precedente, abbiamo definito gli operatori `++` e `++(int)` per la classe `Foo`. L'operatore `++` è l'operatore di incremento prefisso, mentre `++(int)` è l'operatore di incremento postfisso. Gli operatori di decremento sono definiti in modo analogo.

Porocazzo: paragone di due claasi diverse

In C++ possiamo fare l'overloading degli operatori di paragone anche tra due classi diverse. Per fare ciò, dobbiamo definire una funzione globale che prende come argomenti due oggetti delle classi da confrontare.

```
class Foo {
    public:
        int a;
        int b;

        Foo(int a, int b) : a(a), b(b) { }
};

class Bar {
    public:
        int c;
        int d;

        Bar(int c, int d) : c(c), d(d) { }
};
```

continua...

```cpp
bool operator==(const Foo& foo, const Bar& bar) {
    return foo.a == bar.c && foo.b == bar.d;
}

int main() {
    Foo f(42, 69);
    Bar b(42, 69);
    std::cout << (f == b) << std::endl; // Output: 1
}
```

Porocazzo: paragone di due claasi diverse

Nell'esempio precedente, abbiamo definito una funzione globale `operator==` che confronta un oggetto della classe `Foo` con un oggetto della classe `Bar`. La funzione `operator==` prende come argomenti due riferimenti costanti agli oggetti da confrontare.

Organizzazione del codice: header e sorgente

In C++ è possibile organizzare il codice in file header e file sorgente. I file header contengono le dichiarazioni delle classi e delle funzioni, mentre i file sorgente contengono le definizioni delle classi e delle funzioni.

Header file

```
// File: foo.h
class Foo {
    public:
        int a;
        int b;

        Foo(int a, int b);
        bool operator==(const Foo& other) const;
        bool operator!=(const Foo& other) const;
};
```

Source file

```cpp
// File: foo.cpp
#include "foo.h"
#incldue <iostream>

Foo::Foo(int a, int b) : a(a), b(b) { }

bool Foo::operator==(const Foo& other) const {
    return a == other.a && b == other.b;
}

bool Foo::operator!=(const Foo& other) const {
    return !(*this == other);
}
```

Esercizio 1

Si crei un pgm C++ che implementi una classe `Complex` per la gestione dei numeri complessi. Si implementino gli operatori `+`, `-`, `*`, `/`, `==` e `!=` per la classe `Complex` e di seguito si crei un pgm di test.

Esercizio 2

Estendere l'esercizio 1 creando una mini calcolatrice apportando le giuste modiche per gestire operazioni specifiche sui numeri complessi.

“

Madonna che palle sto corso di merda; vado a spararmi un segone.

”

~ Mirkolo 🍻

risolse utilissime per il corso