Confrontiamo le due definizioni

```
ostream& orario::operator<<(ostream& os) const {
  return os << Ore() << ':' << Minuti()
            << ':' << Secondi();
}
```

```
ostream& operator<<(ostream &os,const orario &o) {
  return os << o.Ore() << ':' << o.Minuti()
            << ':' << o.Secondi();
}
```

Differenza: accesso alla parte privata

Altro esempio: somma.

```
orario orario::operator+(orario o) const; // operatore +
orario::orario(int o); // costruttore ad un parametro
```

```
orario t(12,20), s;
s = t + 4;  // OK
```

Invece non funziona

```
s = 4 + t;  // ERRORE: operatore non definito
```

Spiegazione: un metodo di una classe C deve avere come oggetto di invocazione un oggetto di tipo C, non è ammessa una conversione implicita

Operatore + come funzione esterna

```
// operatore + esterno
orario operator+(const orario& t, const orario& s);
orario::orario(int o); // costruttore ad un parametro
```

```
orario t(12,20), s;
s = t + 4;  // OK:conversione del secondo parametro
```

```
s = 4 + t;  // OK:conversione del primo parametro
```

Naturalmente in entrambi i casi funziona

```
s = 4 + 5;  // OK: + tra interi e poi
            // conversione sul risultato
```

Attenzione: non possiamo però definire la funzione esterna come

```
orario operator+(const orario& t, const orario& s)
{
  orario aux;
  aux.sec = (t.sec + s.sec) % 86400;
  return aux;
}
```

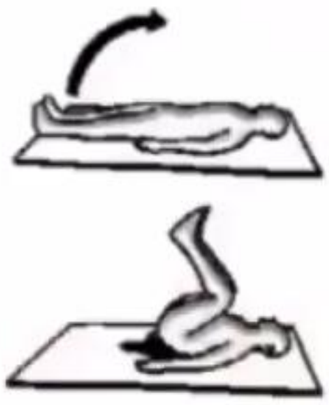
Implementazione di "basso livello" che usa solo l'interfaccia pubblica di orario:

```
orario operator+(const orario& t, const orario& s) {
  int sec = t.Secondi() + s.Secondi();
  int min = t.Minuti() + s.Minuti() + sec / 60;
  sec = sec % 60;
  int ore = t.Ore() + s.Ore() + min / 60;
  min = min % 60;
  ore = ore % 24;
  return orario(ore,min,sec);
}
```

Esercizi

1. **CRUNCH A TERRA**
mani dietro la schiena
20 Rip

2. **CRUNCH A TERRA**
mani sul petto
20 Rip



3. **SOLLEVAMENTO ARTI INFERIORI**
20 rip

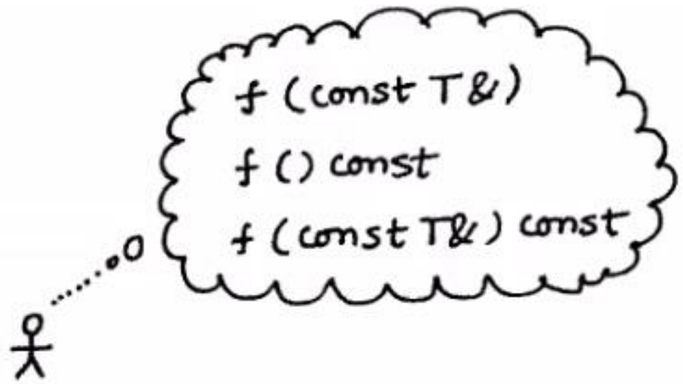
4. FLESSIONE DEL BUSTO
flessione del busto laterale.
20 rip dx 20rip sx

Cosa stampa?

```
class C {
public:
  C() {}
  C(const C& r) {cout << "*";}
};

C f(C a) {
  C b(a);
  C c = b;
  return c;
}

int main() {
  C x;
  C y = f(f(x));
}
```

Esercizio cavillo

LA SETTIMANA ENIGMISTICA

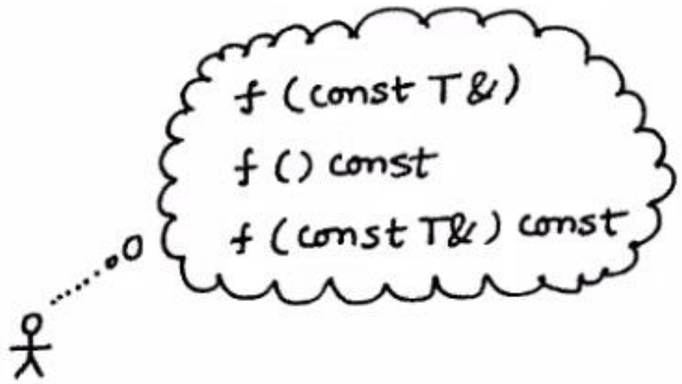



```
class C {
public:
  int x;

  C() { x=8; }

  void f(int& a) const { a=4; } // metodo costante

  void m() { f(x); }            // metodo non costante
};

int main() {
  C c;
  cout << c.x << endl; // stampa: 8
  c.m();
  cout << c.x << endl; // stampa: 4
}
```

const dimenticato?

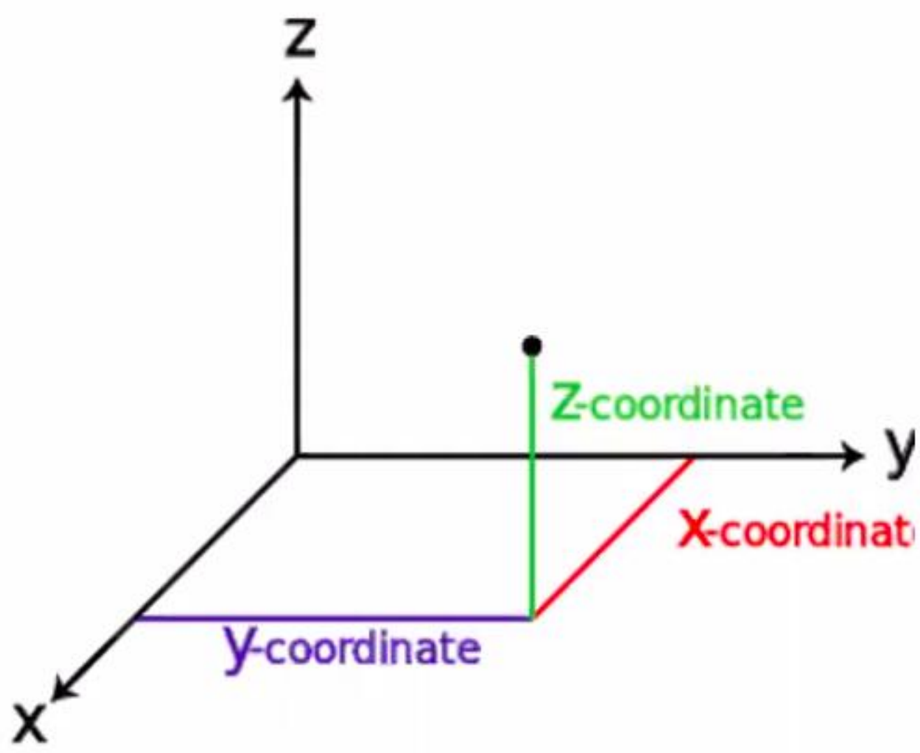
Esercizio cavillo

LA SETTIMANA ENIGMISTICA

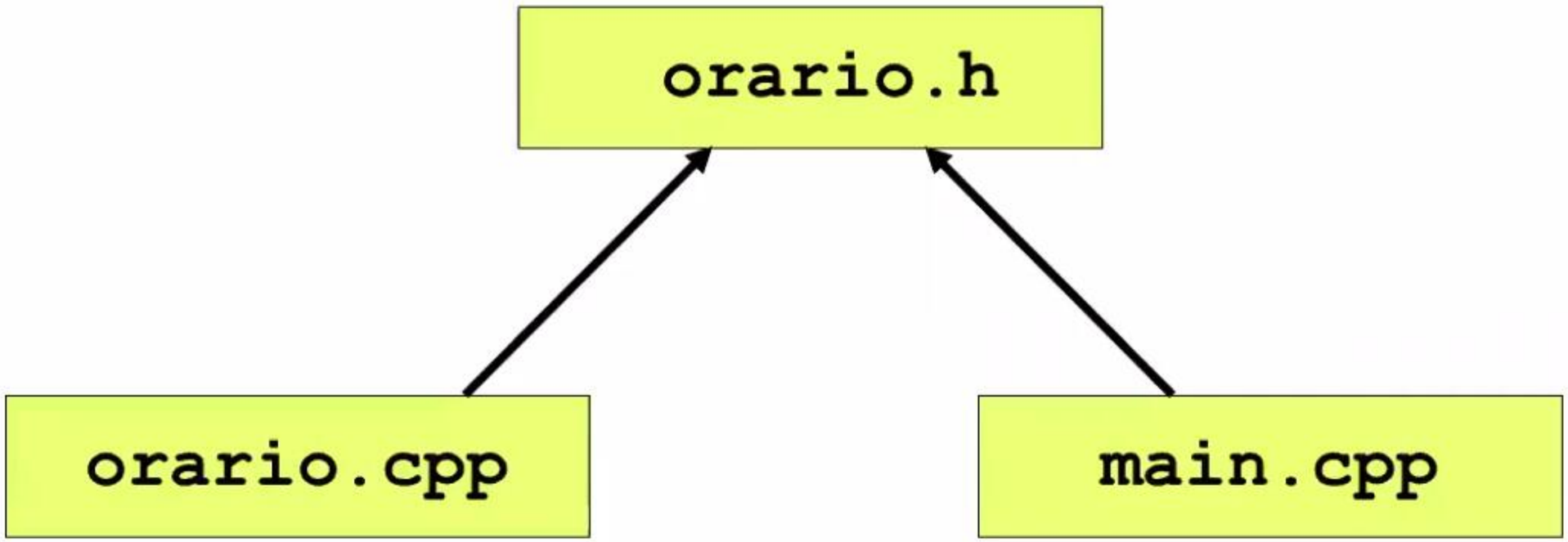



```
class C {
public:
  int x;

  C() { x=8; }

  void f(int& a) const { a=4; }

  void m() const { f(x); } // NON COMPILA: perchè?
};

/*
 g++ error: binding value of type 'const int' to reference to type
'int' drops 'const' qualifier  void m() const { f(x); }
                                                    ^
*/
```

Definire una classe `Point` i cui oggetti rappresentano punti nello spazio (x,y,z). Includere un costruttore di default, un costruttore a tre argomenti che inizializza un punto, selettori delle coordinate cartesiane, un metodo `negate()` che trasforma un punto nel suo negativo, una funzione `norm()` che restituisce la distanza del punto dall'origine, l'overloading degli operatori di somma e di output. Separare interfaccia ed implementazione della classe.



Modularizzazione dei programmi in file




```
g++ -c orario.cpp // compila orario.cpp e non linka
                  // produce il file orario.o
```

- Preprocessore
- Direttive al preprocessore
- **`#include`**: direttiva di inclusione
- **`#define`**: definizione di una macro

C preprocessor

From Wikipedia, the free encyclopedia

The **C preprocessor** or **cpp** is the macro preprocessor for the C and C++ computer programming languages. The preprocessor provides the ability for the inclusion of header files, macro expansions, conditional compilation, and line control.

In many C implementations, it is a separate program invoked by the compiler as the first part of translation.

The language of preprocessor directives is only weakly related to the grammar of C, and so is sometimes used to process other kinds of text files.