

```
const int& f() {return 4;}  
int main() { f(); }
```

g++ ex.cpp

warning: returning reference to temporary [-Wreturn-local-addr]

```
const int& f() {return 4;}  
                ^
```

clang ex.cpp

warning: returning reference to local temporary object [-Wreturn-stack-address]

```
const int& f() {return 4;}  
                ^
```

codice di una slide di ieri che compilando con clang dava dei warning e non tutto okay


Esercizio (prevalentemente di Programmazione)

```
class C {
private:
    int x;
public:
    C(int n = 0) {x=n;}
    C F(C obj) {C r; r.x = obj.x + x; return r;}
    C G(C obj) const {C r; r.x = obj.x + x; return r;}
    C H(C& obj) {obj.x += x; return obj;}
    C I(const C& obj) {C r; r.x = obj.x + x; return r;}
    C J(const C& obj) const {C r; r.x = obj.x + x; return r;}
};

int main() {
    C x, y(1), z(2); const C v(2);
    z=x.F(y); // OK
    //! v.F(y); // ILLEGALE: "passing const C as this discards qualifiers"
    v.G(y); // OK
    (v.G(y)).F(x); // OK
    (v.G(y)).G(x); // OK
    //! x.H(v); // ILLEGALE: "no matching function for call to C::H(const C&)"
    //! x.H(z.G(y)); // ILLEGALE (!!): no matching function for call to C::H(C)
    x.I(z.G(y)); // OK (nota bene!)
    x.J(z.G(y)); // OK
    v.J(z.G(y)); // OK
}
```

Parametro per valore VS riferimento costante

```
class C {  
    int a[1000]; // 4000 bytes  
};  
  
bool byValue(C x) {return true;}  
bool byConstReference(const C& x) {return true;}  
  
int main() {  
    C obj;  
    for(int i=0; i<100000000; i++) byValue(obj); // 3.368 sec  
    for(int i=0; i<100000000; i++) byConstReference(obj); // 0.031 sec  
                                                    // 108x  
}
```





Definire un metodo **OraDiPranzo()** che ritorna sempre uno specifico oggetto della classe **orario** che rappresenta l'orario canonico del pranzo.

Tentativo: un metodo costante **OraDiPranzo()** che ritorna un oggetto della classe **orario**

```
class orario {  
public:  
    orario OraDiPranzo() const;  
    ...  
};
```

```
orario orario::OraDiPranzo() const {  
    return orario(13,15);  
};
```

```
const orario inutile;  
cout << "Si pranza alle "  
    << inutile.OraDiPranzo().Ore() << " e "  
    << inutile.OraDiPranzo().Minuti() << " minuti\n";
```

Static



```
class orario {  
public:  
    static orario OraDiPranzo(); // metodo statico  
    // il modificatore const non ha senso  
    // per un metodo statico perchè il metodo  
    // OraDiPranzo non ha un oggetto di invocazione !  
    ...  
};
```

```
orario orario::OraDiPranzo() {  
    return orario(13,15);  
};
```


Esternamente alla classe si invoca un metodo statico premettendo al nome del metodo il nome della classe e l'operatore di scoping "::".

```
cout << "Si pranza alle "  
      << orario::OraDiPranzo().Ore() << " e "  
      << orario::OraDiPranzo().Minuti() << " minuti\n";
```

Attenzione: nei metodi statici non ha senso il **this**.

NO SENSE

Campi dati statici (o di classe)

Static Object



Dynamic Object

L'inizializzazione dei campi dati statici si deve fare all'esterno della classe ed è **sempre** richiesta.

```
class orario {  
public:  
    ...  
    static int Sec_di_una_Ora;  
    static int Sec_di_un_Giorno;  
  
    ...  
};
```

```
// esternamente alla classe orario  
int orario::Sec_di_una_Ora = 3600;  
int orario::Sec_di_un_Giorno = 86400;
```

Unica copia in memoria dei campi dati statici

Section 9.4.2, Static data members, of the C++ standard states:

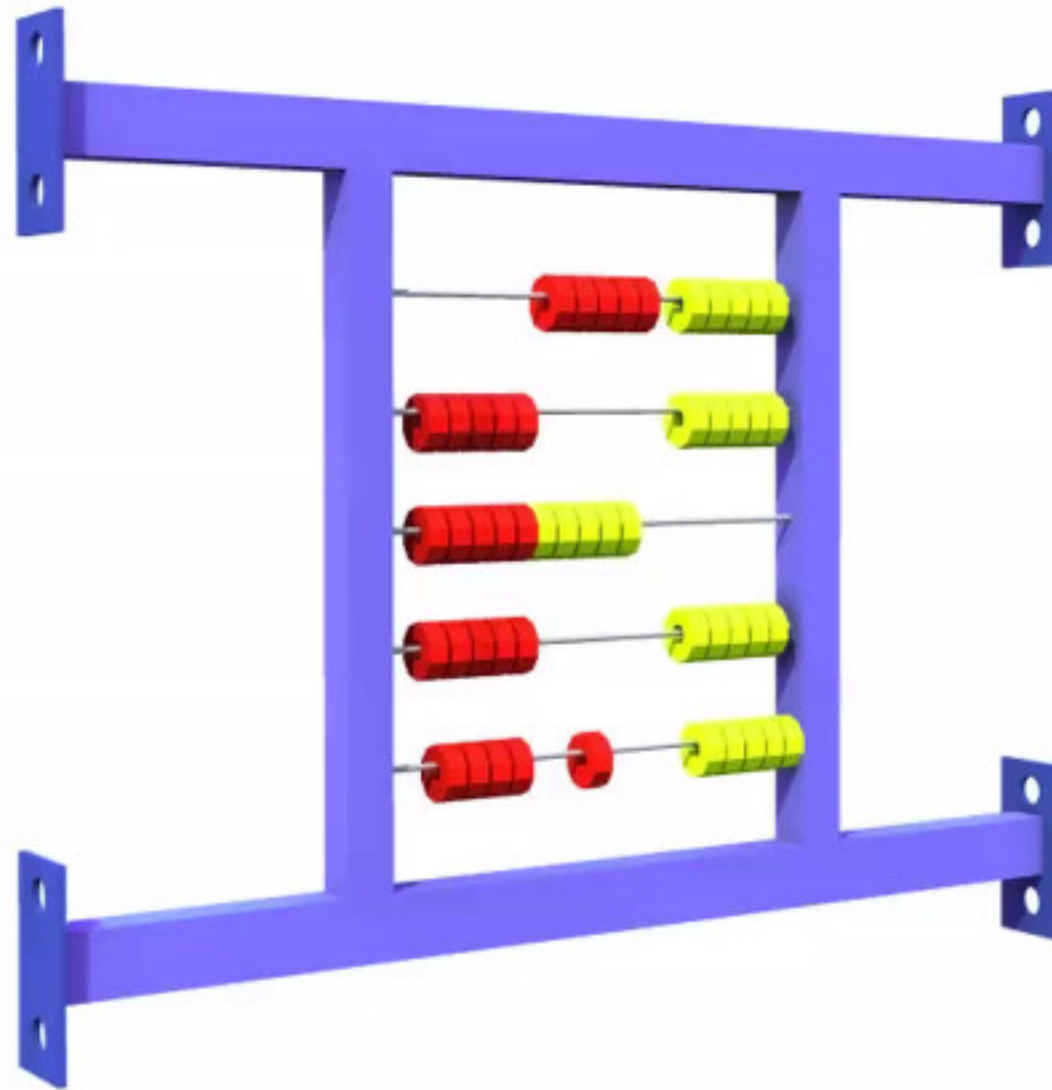
If a `static` data member is of `const` integral or `const` enumeration type, its declaration in the class definition can specify a *const-initializer* which shall be an integral constant expression.

```
class orario {  
public:  
    ...  
    static const int Sec_di_una_Ora = 3600;  
    ...  
};
```



Legale

Esempio: Contare gli oggetti istanziati



Esempio

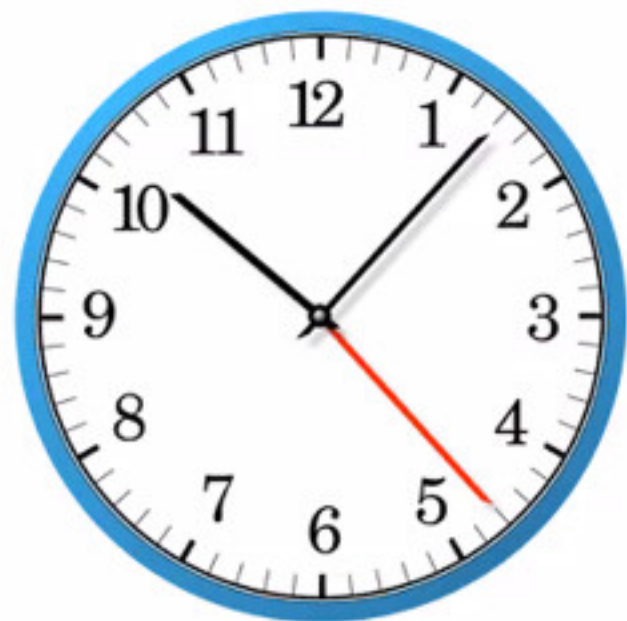
```
#include<iostream>
using namespace std;

class C {
    int dato; // privato
public:
    C(int); // costruttore ad un argomento
    static int cont; // campo dati statico pubblico
};

int C::cont = 0; // inizializzazione campo dati statico

C::C(int n) {cont++; dato=n;} //definizione costruttore

int main(){
    C c1(1), c2(2);
    cout << C::cont; // stampa: 2
}
```



Metodo orario somma(orario)

```
orario orario::Somma(orario o) const {  
    orario aux;  
    aux.sec = (sec + o.sec) % 86400;  
    // Notare che con o.sec si accede ad un campo  
    // dati privato del parametro o  
    return aux;  
}
```

```
int main {  
    ...  
    orario ora(22,45);  
    orario DUE_ORE_E_UN_QUARTO(2,15);  
    ora = ora.Somma(DUE_ORE_E_UN_QUARTO);  
    ...  
}
```




OPERATOR OVERLOADING

"Hello" + "World" = "Hello World"

$1\frac{1}{2} + 2\frac{3}{4} = 4\frac{1}{4}$

```
class orario {  
public:  
    orario operator+(orario) const; // operator è una keyword  
    ....  
};
```

```
orario orario::operator+(orario o) const {  
    orario aux;  
    aux.sec = (sec + o.sec) % 86400;  
    return aux;  
}
```

```
int main {  
    ....  
    orario ora(22,45);  
    orario DUE_ORE_E_UN_QUARTO(2,15);  
    ora = ora + DUE_ORE_E_UN_QUARTO;  
    ....  
}
```

Overloading di un operatore OP

- **operatorOP**

- come metodo oppure come funzione esterna
- se è un metodo allora l'oggetto di invocazione è il primo argomento

Il C++ permette di sovraccaricare circa 40 operatori (unari e binari) tra i quali (lista completa anche su Wikipedia):

```
+ - * / % == != < <= > >= ++ --  
<< >> = -> [] () & new delete
```


Regole per l'overloading degli operatori

RULES

1. YOU CAN....

2. YOU CAN'T...

1) Non si possono cambiare:

- posizione (prefissa/infissa/postfissa)
- numero operandi
- precedenze e associatività

2) Tra gli argomenti deve essere presente almeno un tipo definito dall'utente

3) Gli operatori "=", "[]" e "->" si possono sovraccaricare solo come metodi (interni)

4) **Non si possono** sovraccaricare gli operatori ".", "::", "sizeof", "typeid", i cast e l'operatore condizionale ternario "? :"

5) Gli operatori "=", "&" e "," hanno una versione standard

Operatore condizionale ternario

```
booleanExpr ? expr1 : expr2;
```

```
// ESEMPI
```

```
orario oraApertura = (day == SUNDAY) ? 15 : 9;
```

```
int max(int x, int y) {  
    return x>y ? x : y;  
}
```