Riassunto della classe orario.

```
class orario {
public:
  orario();                 // costruttore di default
  orario(int,int);          // costruttore ore-minuti
  orario(int,int,int);      // costruttore ore-minuti-secondi
  int Ore();                // selettore delle ore
  int Minuti();             // selettore dei minuti
  int Secondi();            // selettore dei secondi
private:
  int sec;                  // campo dati
};
```

Costruttore standard

```
class orario {
public:
// nessuna definizione di costruttori
   ...
};

orario o; // OK: viene invocato il
          // costruttore standard di default
```


```
class orario {
public:
   orario(int,int); // solo costruttore ore-minuti
   ...
};

orario o; // Errore: manca un costruttore di default
```

Altri esempi.

```
orario adesso(11,55);    // costruttore ore-minuti
orario copia;            // costruttore di default
copia = adesso;          // assegnazione
orario copia1 = adesso;  // costruttore di copia
                         // analogo a int x = y;
orario copia2(adesso);   // costruttore di copia
                         // analogo a int x(y);
```

Il **costruttore di copia** crea un nuovo oggetto **`copia1`** in cui copia, campo dati per campo dati, l'oggetto **`adesso`**.
L'**assegnazione** assegna il valore di ogni campo dati dell'oggetto **`adesso`** al corrispondente campo dati dell'oggetto preesistente **`copia`**.

```
C(const C&)
C& operator=(const C&)
```

Costruttori come convertitori di tipo

costruttore `C(T)`

conversione implicita `T` $\Rightarrow$ `C`

```
orario s;
s = 8;
```

Con l'assegnazione **`s = 8;`** succede che:
1) viene invocato il costruttore **`orario(int)`** con parametro attuale 8 che crea un oggetto anonimo temporaneo della classe **`orario`**
2) l'oggetto temporaneo viene assegnato all'oggetto **`s`**
3) viene "deallocato" l'oggetto temporaneo

```
orario s,t;
s = 8;          // OK: equivale a s = orario(8)
t = 8+12;       // OK: equivale a t = orario(8+12)
```

Argomenti di default nelle funzioni


```
void F(double x, int n = 3, string s); // ILLEGALE!

void G(double x, int n = 3, string s = "ciao"); // OK

G(3.2); // OK: G(3.2,3,"ciao");
G();    // NO
G(3,3); // OK: G(3,3,"ciao");
G(3,3,"pippo"); // OK: G(3,3,"pippo");
```

```
orario::orario(int o = 0, int m = 0, int s = 0)
{
  if (o < 0 || o > 23 || m < 0 || m > 59 ||
      s < o || s > 59) sec = 0;
  else sec = o * 3600 + m * 60 + s;
};
```

Costruttore a 0, 1, 2 e 3 parametri.

Agisce anche come convertitore di tipo da **`int`** a **`orario`**.

```
class orario {
public:
  // costruttore a tre parametri con argomenti di default
  orario(int =0,int =0,int =0);
  int Ore();                    // selettore delle ore
  int Minuti();                 // selettore dei minuti
  int Secondi();                // selettore dei secondi
private:
  int sec;
};
```

```
// attenzione alla sintassi
orario::orario(int o, int m, int s) {
  if (o < 0 || o > 23 || m < 0 || m > 59 ||
      s < 0 || s > 59) sec = 0;
  else sec = o * 3600 + m * 60 + s;
}

int orario::Ore() { return sec / 3600; }
int orario::Minuti() { return (sec / 60) % 60; }
int orario::Secondi() { return sec % 60; }
```

Esempi

```
#include<iostream>
#include "orario.cpp"
using namespace std::cout; using namespace std::endl;

int main() {
  orario s;
  s = 6; // equivale a: s = orario(6,0,0);
  cout << s.Ore() << ':' << s.Minuti() << ':'
       << s.Secondi() << endl;
  orario t = 5+2*2; // equivale a: orario t = orario(5+2*2,0,0);
  cout << t.Ore() << ':' << t.Minuti() << ':'
       << t.Secondi() << endl;
  orario r(12,45); // equivale a: orario r(12,45,0);
  cout << r.Ore() << ':' << r.Minuti() << ':'
       << r.Secondi() << endl;
  orario a; // equivale a: orario a(0,0,0);
  cout << a.Ore() << ':' << a.Minuti() << ':'
       << a.Secondi() << endl;
  orario b(7); // equivale a: orario b(7,0,0);
  cout << b.Ore() << ':' << b.Minuti() << ':'
       << b.Secondi() << endl;
}
```

Operatori espliciti di conversione

In una classe C, operatore di conversione $C \Rightarrow T$

```
class orario {
public:
   operator int() {return sec;} // orario ⇒ int
   ...
};

orario o(14,37);
int x = o;     // OK: viene richiamato
               // implicitamente il metodo int() su o
```

Parola chiave `explicit`


```
class orario {
public:
   explicit orario(int); // costruttore esplicito
                         // con un solo parametro
   ...
};

orario o = 8; // Errore: non viene richiamato
              // implicitamente il costruttore
              // orario(8)
```

Aggiungiamo a `orario` due nuovi metodi pubblici

```
class orario {
public:
   ...
   orario UnOraPiuTardi();
   void  AvanzaUnOra();
private:
   ...
};
```

```
orario orario::UnOraPiuTardi() {
   orario aux;
   aux.sec = (sec + 3600) % 86400;
   return aux;
};
```

```
void orario::AvanzaUnOra() {
   sec = (sec + 3600) % 86400;
};
```

```
orario mezzanotte;
cout << mezzanotte.Ore();    // stampa 0
orario adesso(15);
cout << adesso.Ore();        // stampa 15
adesso = mezzanotte.UnOraPiuTardi();
cout << adesso.Ore();        // stampa 1
cout << mezzanotte.Ore();    // stampa 0
mezzanotte.AvanzaUnOra();
cout << mezzanotte.Ore();    // stampa 1
```

Metodi e oggetti di invocazione costanti