Tipi e template impliciti in template di classe

```
template <class T>
class C {
public:
  class X {};          // template di classe annidata associato
                       // potrebbe usare il parametro di tipo T

  class D {            // template di classe annidata associato
  public:              // usa il parametro di tipo T
    T x;
  };

  template <class U>
  class E {            // template di classe annidata
  public:              // non associato
    T x;               // usa T del template contenitore
    U y;               // usa il suo parametro di tipo U
    void fun1() {return;}
  };

  template <class U>
  void fun2() {        // template di metodo di istanza
    T x; U y; return; // non associato
  }
};
```

Tipi e template impliciti in template di classe

```
template <class T>
  // C<T>::D è un uso di un tipo che
  // effettivamente dipende dal parametro T
void templateFun(typename C<T>::D d) {

  // C<T>::X è un uso di un tipo che
  // comunque dipende dal parametro T
  typename C<T>::X x;

  // C<T>::D è un uso di un tipo che
  // effettivamente dipende dal parametro T
  typename C<T>::D d2 = d;

  // (1) E<int> è un uso del template di classe
  //     annidata che dipende dal parametro T
  // (2) C<T>::E<int> è un uso di un tipo
  //     che dipende dal parametro T
  typename C<T>::template E<int> e;
  e.fun1();

  // c.fun2<int> è un uso del template di funzione
  // che dipende dal parametro T
  C<T> c;
  c.template fun2<int>();
}
```

Forward declarations for templates (check your g++)

```
// dichiarazione incompleta (alias forward declaration)
template<class T> class Queue;

// dichiarazione del template di funzione operator<< <T>
template<class T> std::ostream& operator<<(std::ostream& os,const Queue<T>&);

// dichiarazione incompleta non permessa
// template<class T> class Queue<T>::QueueItem;

// dichiarazione non necessaria
// template<class T> ostream& operator<<(ostream& os,const typename Queue<T>::QueueItem&);

template<class T> class Queue {
  // NON agisce da dichiarazione di template di funzione
  friend std::ostream& operator<< <T> (std::ostream& os, const Queue<T>&);
private:
  class QueueItem {
    friend class Queue<T>;
    friend std::ostream& operator<< <T> (std::ostream& os, const Queue<T>&);
    // agisce da dichiarazione di template di funzione
    friend std::ostream& operator<< <T> (std::ostream& os, const typename Queue<T>::QueueItem&);
  private:
    T info;
    QueueItem* next;
  };
  QueueItem *primo, *ultimo;
public:
  // ...
};

// definizione del template di funzione operator<< <T>
template<class T> std::ostream& operator<< (std::ostream& os, const Queue<T>& q) {
  // ...
  return os;
}

// definizione del template di funzione operator<< <T>
template<class T> std::ostream& operator<< (std::ostream& os, const typename Queue<T>::QueueItem& qi) {
  // ...
  return os;
}
```