Compilazione dei Template

Q: Come analizzare se un template compilerà?

A: Verificare:

1. **Corrispondenza dei Parametri**:

```
template<class T1, class T2 = Z, int k = 1>
class C { ... };

// Verifica istanziazioni:
C<int>         // OK: T2=Z, k=1 (default)
C<int,double>  // OK: k=1 (default)
C<int,double,3>// OK: tutti specificati
```

2. **Accesso ai Membri**:

```
template<class T>
class D {
    void f() { C<T,T> c; c.x; }  // Compila solo se:
    // 1. C<T,T> è valido
    // 2. x è accessibile (public o friend)
};
```

3. **Friend Declarations**:

```
template<class T> class D;  // Dichiarazione forward necessaria

template<class T1, class T2>
class C {
    friend class D<T1>;  // D può accedere ai privati di C
};
```

Pattern di Catch ed Eccezioni

Q: Come funzionano i pattern con catch e template?

A:

1. **Ordine dei Catch**:

```
template<class T>
void Fun(const T& ref) {
    try { throw ref; }
    catch(const DerivedMost& d) { ... }  // Prima i più specifici
    catch(const Base& b) { ... }         // Poi i più generali
    catch(...) { ... }                   // Catch-all alla fine
}
```

2. **Type Deduction**:

```
Fun(c);       // T dedotto dal tipo dell'argomento
Fun<Base>(d); // T specificato esplicitamente
```

3. **Controllo di Compilazione**:

```
// Non compila se:
Fun<Derived>(base);    // Tipo specificato incompatibile
Fun<int>(string_obj);  // Tipi non correlati
```

Parametri Template e Friend

Q: Come gestire parametri template e friend?

A:

1. **Default Parameters**:

```
template<class T1, class T2 = Z, int k = 1>
class C {
    // T2 e k hanno valori di default
    // Istanziazioni valide:
    // C<int>
    // C<int,double>
    // C<int,double,42>
};
```

2. **Friend Access**:

```
template<class T> class D;  // Forward declaration

template<class T1, class T2>
class C {
    friend class D<T1>;  // D<T1> può accedere ai privati
private:
    T1 t1;
```

```
    T2 t2;
};
```

3. **Type Safety**:

```
template<class T>
class D {
    void f() {
        C<T,T> c;     // OK se T è un tipo valido per entrambi i parametri
        C<int,T> c;   // OK se T è compatibile come secondo parametro
    }
};
```

Pattern di Errori Comuni

Q: Quali sono i pattern di errori più comuni?

A:

1. **Mancata Forward Declaration**:

```
// Errore:
class C {
    friend class D<T>;  // D non dichiarato
};

// Corretto:
template<class T> class D;  // Forward declaration prima
class C { ... };
```

2. **Accesso a Membri**:

```
template<class T>
class D {
    void f() {
        C<T> c;
        c.private_member;  // Errore se non friend
    }
};
```

3. **Type Parameter Mismatch**:

```
template<class T>
void f() {
```

```
    C<T,T> c;  // Errore se T non valido per entrambi i parametri
}
```

Best Practices

1. **Organizzazione del Codice**:
   - Dichiarare forward declarations all'inizio
   - Organizzare friend declarations subito dopo
   - Mantenere una chiara gerarchia di classi
2. **Type Safety**:
   - Verificare compatibilità dei tipi
   - Documentare requisiti sui tipi template
3. **Debug**:
   - Testare con tipi semplici prima
   - Verificare casi limite
   - Controllare requisiti di friend e accesso