

# CORSO GOLIA : RUNTIME > SO ROMPI

Siano A, B, C e D quattro diverse classi polimorfe. Si considerino le seguenti definizioni.

```
template <class X, class Y>
X* fun(X* p) { return dynamic_cast<Y*>(p); }

main() {
    C c; fun<A,B>(&c);
    if( fun<A,B>(new C()) == 0 ) cout << "Bjarne ";
    if( dynamic_cast<C*>(new B()) == 0 ) cout << "Stroustrup";
    A* p = fun<D,B>(new D());
}
```

Si supponga che:

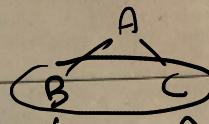
1. il main() compili correttamente ed esegua senza provocare errori a run-time;
2. l'esecuzione del main() provochi in output su cout la stampa Bjarne Stroustrup.

In tali ipotesi, per ognuna delle relazioni di sottotipo  $T_1 \leq T_2$  nelle seguenti tabelle segnare con una croce l'entrata

- (a) "Vero" per indicare che  $T_1$  sicuramente è sottotipo di  $T_2$ ;
- (b) "Falso" per indicare che  $T_1$  sicuramente non è sottotipo di  $T_2$ ;
- (c) "Possibile" altrimenti, ovvero se non valgono né (a) né (b).

	Vero	Falso	Possibile
$A \leq B$			
$A \leq C$			
$A \leq D$			
$B \leq A$			
$B \leq C$			
$B \leq D$			

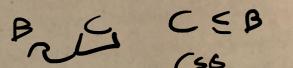
	Vero	Falso	Possibile
$C \leq A$			
$C \leq B$			
$C \leq D$			
$D \leq A$			
$D \leq B$			
$D \leq C$			



POSSIBILE  
=  
STESSO C.V.  
G ELENCAZ

```
template <class A, class B>
A* fun(A* p) { return dynamic_cast<B*>(p); }

main() {
    C c; fun<A,B>(&c);
    if( fun<A,B>(new C()) == 0 ) cout << "Bjarne ";
    if( dynamic_cast<C*>(new B()) == 0 ) cout << "Stroustrup";
    A* p = fun<D,B>(new D());
}
```



$C \leq B$   
CSE  
VA B SNO

Siano A, B, C e D quattro diverse classi polimorfe. Si considerino le seguenti definizioni.

```
template <class A, class B>
A* fun(A* p) { return dynamic_cast<B*>(p); }

main() {
    C c; fun<A,B>(&c);
    if( fun<A,B>(new C()) == 0 ) cout << "Bjarne ";
    if( dynamic_cast<C*>(new B()) == 0 ) cout << "Stroustrup";
    A* p = fun<D,B>(new D());
}
```

Si supponga che:

1. il main() compili correttamente ed esegua senza provocare errori a run-time;
2. l'esecuzione del main() provochi in output su cout la stampa Bjarne Stroustrup.

In tali ipotesi, per ognuna delle relazioni di sottotipo  $T_1 \leq T_2$  nelle seguenti tabelle segnare con una croce l'entrata

↓ SOTTO PPI

Siano A, B, C e D quattro diverse classi polimorfe. Si considerino le seguenti definizioni.

```
template <class A, class B>
A* fun(A* p) { return dynamic_cast<B*>(p); }
```

↑ SFATICO

```
main() {
    C c; fun<A,B>(&c);
    if( fun<A,B>(new C()) == 0 ) cout << "Bjarne ";
    if( dynamic_cast<C*>(new B()) == 0 ) cout << "Stroustrup";
    A* p = fun<D,B>(new D());
}
```

$A * (.. B) \rightarrow B \leq A$

$C \not\leq B$  [  $B \sim C$  ]  
 $B \not\leq C$  [  $C \sim B$  ]

Si supponga che:

1. il main() compili correttamente ed esegua senza provocare errori a run-time;
2. l'esecuzione del main() provochi in output su cout la stampa Bjarne Stroustrup.

In tali ipotesi, per ognuna delle relazioni di sottotipo  $T_1 \leq T_2$  nelle seguenti tabelle segnare con una croce l'entrata

Siano A, B, C e D quattro diverse classi polimorfe. Si considerino le seguenti definizioni.

```
template <class D, class B>
D* fun(D* p) { return dynamic_cast<B*>(p); }
```

CORRETTO?  
↓ NO!

```
main() {
    C c; fun<A,B>(&c);
    if( fun<A,B>(new C()) == 0 ) cout << "Bjarne ";
    if( dynamic_cast<C*>(new B()) == 0 ) cout << "Stroustrup";
    A* p = fun<D,B>(new D());
}
```

$D \leq B$   
 $D \leq A$

$C \not\leq B$ ,  $C \leq A$   
 $B \not\leq C$

$\rightarrow TS = A \rightarrow D \leq A$

Si supponga che:

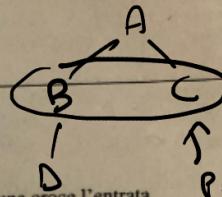
1. il main() compili correttamente ed esegua senza provocare errori a run-time;
2. l'esecuzione del main() provochi in output su cout la stampa Bjarne Stroustrup.

Siano A, B, C e D quattro diverse classi polimorfe. Si considerino le seguenti definizioni.

```
template <class X, class Y>
X* fun(X* p) { return dynamic_cast<Y*>(p); }

main() {
    C c; fun<A,B>(&c); → A ≠ B, B ⊆ A
    if( fun<A,B>(new C()) == 0 ) cout << "Bjarne ";
    if( dynamic_cast<C*>(new B()) == 0 ) cout << "Stroustrup";
    A* p = fun<D,B>(new D());
} → D ⊆ A
```

corretto?  
IF(P) ...  
6 no



Si supponga che:

- 1. il main() compili correttamente ed esegua senza provocare errori a run-time;
- 2. l'esecuzione del main() provochi in output su cout la stampa Bjarne Stroustrup.

In tali ipotesi, per ognuna delle relazioni di sottotipo  $T_1 \leq T_2$  nelle seguenti tabelle segnare con una croce l'entrata

- (a) "Vero" per indicare che  $T_1$  sicuramente è sottotipo di  $T_2$ ;  
(b) "Falso" per indicare che  $T_1$  sicuramente non è sottotipo di  $T_2$ ;  
(c) "Possibile" altrimenti, ovvero se non valgono né (a) né (b).

→ INVIO

	Vero	Falso	Possibile
$A \leq B$			
$A \leq C$		✗	
$A \leq D$		✗	
$B \leq A$	✗		
$B \leq C$			✗
$B \leq D$		✗	

→ POSSIBILE  
STESO UN.  
G ELLA

	Vero	Falso	Possibile
$C \leq A$	✗		
$C \leq B$			✗
$C \leq D$			✗
$D \leq A$	✗		
$D \leq B$	✗		
$D \leq C$		✗	

### Esercizio 3

```
class A {
protected:
    virtual void r() {cout << "A::r ";}
public:
    virtual void g() const {cout << "A::g ";}
    virtual void f() {cout << "A::f "; g(); r();}
    void m() {cout << "A::m "; g(); r();}
    virtual void k() {cout << "A::k "; r(); m();}
    virtual A* n() {cout << "A::n "; return this;}
};

class C: public A {
protected:
    void r() {cout << "C::r ";}
public:
    virtual void g() {cout << "C::g ";}
    void m() {cout << "C::m "; g(); r();}
    void k() const {cout << "C::k "; k();}
};

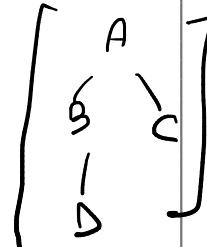
class B: public A {
public:
    virtual void g() const {cout << "B::g ";}
    virtual void m() {cout << "B::m "; g(); r();}
    void k() {cout << "B::k "; A::n();}
    A* n() {cout << "B::n "; return this;}
};

class D: public B {
protected:
    void r() {cout << "D::r ";}
public:
    B* n() {cout << "D::n "; return this;}
    void m() {cout << "D::m "; g(); r();}
};

A* p1 = new D(); A* p2 = new B(); A* p3 = new C(); B* p4 = new D(); const A* p5 = new C();
```

$P1 \rightarrow K()$

$\{TS = A \quad PD = D\}$



Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- NON COMPILA se la compilazione dell'istruzione provoca un errore;
- ERRORE RUN-TIME se l'istruzione compila correttamente ma la sua esecuzione provoca un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva NESSUNA STAMPA.

CONS GNA:  $[P1 \rightarrow K()] \quad A::K / D::R / A::m$

$m \rightarrow$  NON  
VISUAL

$A::g /$   
 $A::m$

$A \propto P^2 = \text{new } B()$

### Esercizio 3

```

class A {
protected:
    virtual void r() {cout<<" A::r ";}
public:
    virtual void g() const {cout <<" A::g ";}
    virtual void f() {cout <<" A::f "; g(); r();}
    void m() {cout <<" A::m "; g(); r();}
    virtual void k() {cout <<" A::k "; r(); m();}
    virtual A* n() {cout <<" A::n "; return this;}
};

class C: public A {
protected:
    void r() {cout <<" C::r ";}
public:
    virtual void g() {cout <<" C::g ";}
    void m() {cout <<" C::m "; g(); r();}
    void k() const {cout <<" C::k "; k();}
};

```

STRATEGY  
MANAGER! ~~ORP~~

```

class B: public A {
public:
    virtual void g() const {cout <<" B::g ";}
    virtual void m() {cout <<" B::m "; g(); r();}
    void k() {cout <<" B::k "; A::n();}
    A* n() {cout <<" B::n "; return this;}
};

class D: public B {
protected:
    void r() {cout <<" D::r ";}
public:
    B* n() {cout <<" D::n "; return this;}
    void m() {cout <<" D::m "; g(); r();}
};

A* p1 = new D(); A* p2 = new B(); A* p3 = new C(); B* p4 = new D(); A* p5 = new C();

```

STATIC-CAT (C<sup>++</sup>) (P2) → K

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
  - **ERRORE RUN-TIME** se l'istruzione compila correttamente ma la sua esecuzione provoca un errore a run-time;
  - se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

*'gut* =

```
(static_cast<C*>(p2))->k();
```

ଓঠে ন  
বালু

### Esercizio 3

```

class A {
protected:
    virtual void r() {cout << " A::r ";}
public:
    virtual void g() const {cout << " A::g ";}
    virtual void f() {cout << " A::f "; g(); r();}
    void m() {cout << " A::m "; g(); r();}
    virtual void k() {cout << " A::k "; r(); m();}
    virtual A* n() {cout << " A::n "; return this;}
};

class B: public A {
public:
    virtual void g() const {cout << " B::g ";}
    virtual void m() {cout << " B::m "; g(); r();}
    void k() {cout << " B::k "; A::n();}
    A* n() {cout << " B::n "; return this;}
};

class C: public A {
protected:
    void r() {cout << " C::r ";}
public:
    virtual void g() {cout << " C::g ";}
    void m() {cout << " C::m "; g(); r();}
    void k() const {cout << " C::k "; k();}
};

class D: public B {
protected:
    void r() {cout << " D::r ";}
public:
    B* n() {cout << " D::n "; return this;}
    void m() {cout << " D::m "; g(); r();}
};

A* p1 = new D(); A* p2 = new B(); A* p3 = new C(); B* p4 = new D(); const A* p5 = new C();



```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;

## UB → UNSTRUCTURED BEHAVIOR

COSA → COMPILA .  
SImpre NON COMPILA (NC)

UB / BRT

susgu uno

U3 → CORROSION AND WEAR

## PERFORMANCE DATA STANDARDS ...

[ DYNAMIC-CAST (C $\Rightarrow$  C<sub>B</sub>) ]  $\rightarrow$  NON  
SARROGNO

$\downarrow$   
 - A CLASS PARENT  
 A  
 $[B] [C]$  - B/C SONO NULL  
 $\Rightarrow$  LA CONV. COSTRA?  
 $(NC) \rightarrow \underline{0}$

ERRORS RUNTIME ]  $\rightarrow$  CPP  $\rightarrow$   $\begin{cases} g++ \\ ss.cpp \end{cases}$   
 $\downarrow$   
 $> CC: K() C :: K() \dots$  (SSGU)  
 (ERRORS RUNTIME)  
 $\dots$   
 $\text{"STAMP INNUTA!"}$

K()  $\rightarrow$  EXP

CASE  
 PARTE CALCOLO ]  $\rightarrow$  RETURN THIS  
 PART CALCULUS  
 $\rightarrow NC$

A  
~~VIRG~~ void  
 $\downarrow$   
 $B \neq NC$   
 RETURN THIS  
 $B(A)$

$\{B::NC \quad A::K()\} \rightarrow THIS$   
 OCCORRE IN QUESTO CASO!  
 $BS = A \quad TD = B$

---