COMPILA / NC ?

```
C f(C& x) {return x;}
C& g() const {return *this;}
C h() const {return *this;}
C* m() {return this;}
C* n() const {return this;}
void p() {}
void q() const {p();}
void p() {}
static void r(C *const x) {x->p();}
void s(C *const x) const {*this = *x;}
static C& t() {return C();}
static C *const u(C& x) {return &x;}
```

→ C

→ NC = CONST-CORRECTNESS

→ C

→ NC = CONST-CORRECT

→ NC → VOID P() CONST (OR) VOID Q() CONST

NON TOCCA CONST CORRETTEZZA DI P()

→ C

→ NC

→ NC

→ C

Si assuma che A, B, C, D siano quattro classi polimorfe. Si consideri il seguente main().

```
main() {
 A a; B b; C c; D d;
 cout << (dynamic_cast<D*>(&c) ? "0 " : "1 ");
 cout << (dynamic_cast<B*>(&c) ? "2 " : "3 ");
 cout << (!(dynamic_cast<C*>(&b)) ? "4 " : "5 ");
 cout << (dynamic_cast<B*>(&a) || dynamic_cast<C*>(&a)  ? "6 " : "7 ");
 cout << (dynamic_cast<D*>(&b) ? "8 " : "9 ");
}
```

Si supponga che tale main() compili ed esegua correttamente. Disegnare i diagrammi di **tutte** le possibili gerarchie per le classi A, B, C, D tali che l'esecuzione del main() provochi la stampa: 0 3 4 6 8.

- 0 (?) → A B C D

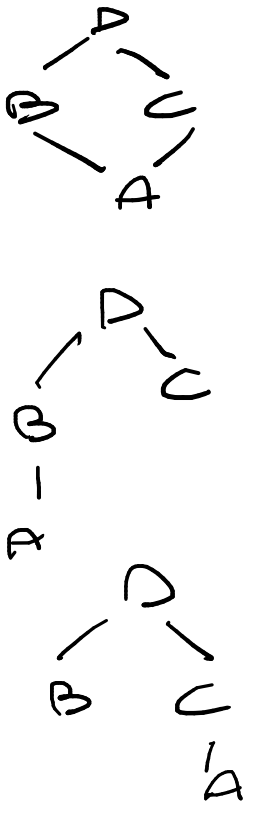
  → C ⊆ D (V)

- 3 (?) → B ⊄ C (V)

- 4 (?) → B ⊄ C (V)

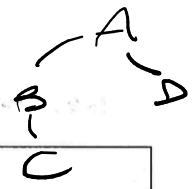
- 6 (?) → A ⊆ B, A ⊆ C (V)

- 8 (?) → B ⊆ D



Esercizio Tipi

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




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

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

Si supponga che:

1. il `main()` compili correttamente ed esegua senza provocare undefined behaviour;
2. l'esecuzione del `main()` provochi in output su `cout` la stampa `Bjarne Stroustrup`.

In tali ipotesi, per ognuna delle relazioni di sottotipo `T1`≤`T2` nelle seguenti tabelle segnare con una croce l'entrata

(a) "Vero" per indicare che `T1` **sicuramente** è sottotipo di `T2`;
(b) "Falso" per indicare che `T1` **sicuramente non** è sottotipo di `T2`;
(c) "Possibile" **altrimenti**, ovvero se non valgono né (a) né (b).

Si considerino le seguenti definizioni.

```
class B {
private:
  list<double>* ptr;
  virtual void m() =0;
};

class D: virtual public B {
private:
  int x;
};

class E: public C, public D {
private:
  vector<int*> v;
public:
  void m() {}
  // ridefinizione del costruttore di copia di E
};
```

```
class C: virtual public B {};
```

Ridefinire il costruttore di copia di `E` in modo tale che il suo comportamento coincida con quello del costruttore di copia standard di `E`.

```
class Z {
 public: Z(int x) {}
};
```

```
class A {
 public:
  void f(int) {cout << "A::f(int) "; f(true);}
  virtual void f(bool) {cout <<"A::f(bool) ";}
  virtual A* f(Z) {cout <<"A::f(Z) "; f(2); return this;}
  A() {cout <<"A() "; }
};
```

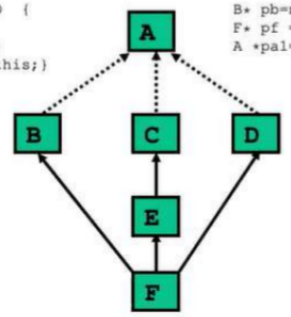
```
class B: virtual public A {
 public:
  void f(const bool&){cout<< "B::f(const bool&) ";}
  void f(const int&){cout<< "B::f(const int&) ";}
  virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
  virtual ~B() {cout << "~B ";}
  B() {cout <<"B() "; }
};
```

```
class C: virtual public A {
 public:
  C* f(Z){cout <<"C::f(Z) "; return this;}
  C() {cout <<"C() "; }
};
```

```
class D: virtual public A {
public:
  virtual void f(bool) const {cout <<"D::f(bool) ";}
  A* f(Z) {cout << "D::f(Z) "; return this;}
  ~D() {cout <<"~D ";}
  D() {cout <<"D() ";}
};
```

```
class E: public C {
 public:
  C* f(Z){cout <<"E::f(Z) "; return this;}
  ~E() {cout <<"~E ";}
  E() {cout <<"E() ";}
};
```

```
class F: public B, public E, public D  {
 public:
  void f(bool){cout<< "F::f(bool) ";}
  F* f(Z){cout <<"F::f(Z) "; return this;}
  F() {cout <<"F() "; }
  ~F() {cout <<"~F ";}
};
```




```
B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pbl= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```