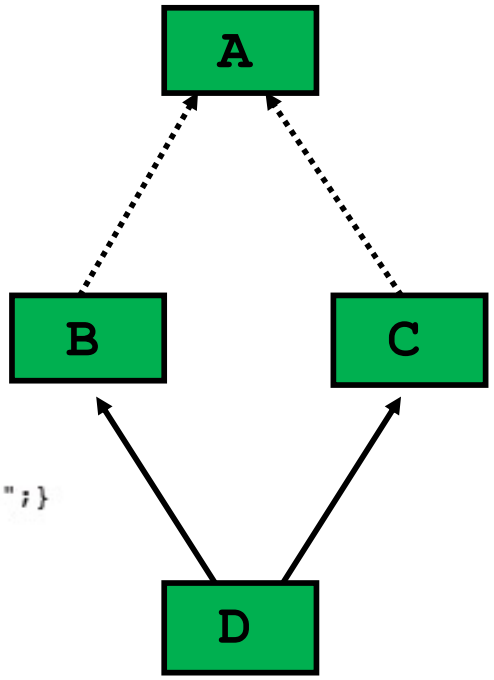
```
class Z {
public:
  Z() {cout << "Z() ";}
  Z(const Z& x) {cout << "Zc ";}
};

class A {
private:
  Z w;
public:
  A() {cout << "A() ";}
  A(const A& x) {cout << "Ac ";}
};

class B: virtual public A {
private:
  Z z;
public:
  B() {cout << "B() ";}
  B(const B& x) {cout << "Bc ";}
};

class C: virtual public A {
private:
  Z z;
public:
  C() {cout << "C() ";}
};

class D: public B, public C {
public:
  D() {cout << "D() ";}
  D(const D& x): C(x) {cout << "Dc ";}
};
```

Cosa stampa?

```
D d1;
```

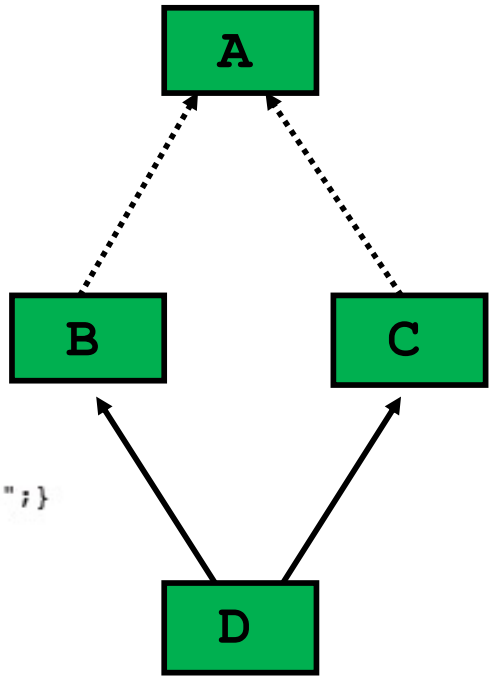
```
Z() A() Z() B() Z() C() D()
```



```
class Z {
public:
  Z() {cout << "Z() ";}
  Z(const Z& x) {cout << "Zc ";}
};

class A {
private:
  Z w;
public:
  A() {cout << "A() ";}
  A(const A& x) {cout << "Ac ";}
};

class B: virtual public A {
private:
  Z z;
public:
  B() {cout << "B() ";}
  B(const B& x) {cout << "Bc ";}
};

class C: virtual public A {
private:
  Z z;
public:
  C() {cout << "C() ";}
};

class D: public B, public C {
public:
  D() {cout << "D() ";}
  D(const D& x): C(x) {cout << "Dc ";}
};
```

Cosa stampa?

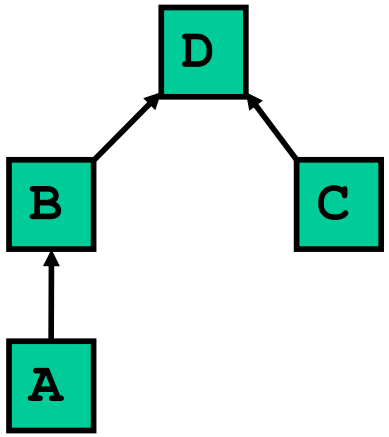
```
D d2=d1;
Z() A() Z() B() Zc Dc
```



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`.



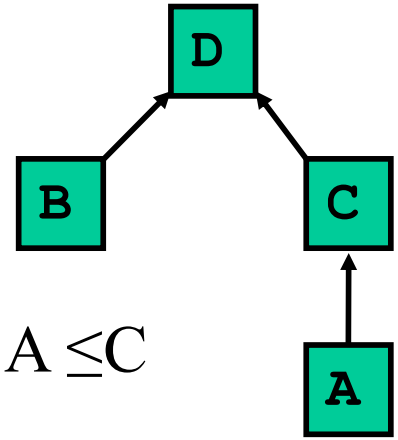
$C \leq D$

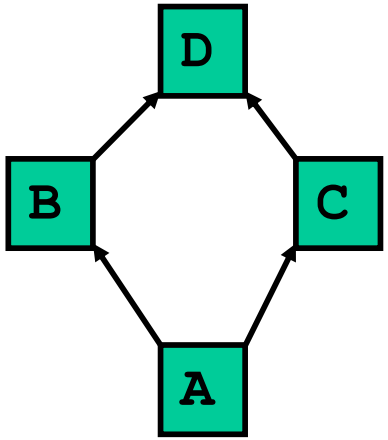
$C \not\leq B$

$B \not\leq C$

$A \leq B$ or $A \leq C$

$B \leq D$





Esercizio

Sia `B` una classe polimorfa e sia `C` una sottoclasse di `B`. Definire una funzione `int Fun(const vector<B*>& v)` con il seguente comportamento: sia `v` non vuoto e sia `T*` il tipo dinamico di `v[0]`; allora `Fun(v)` ritorna il numero di elementi di `v` che hanno un tipo dinamico `T1*` tale che `T1` è un sottotipo di `C` diverso da `T`; se `v` è vuoto deve quindi ritornare 0. Ad esempio, il seguente programma deve compilare e provocare le stampe indicate.

```
#include<iostream>
#include<typeinfo>
#include<vector>
using namespace std;

class B {public: virtual ~B()
class C: public B {};
class D: public B {};
class E: public C {};

int Fun(vector<B*> &v){...}

main() {
  vector<B*> u, v, w;
  cout << Fun(u); // stampa 0
  B b; C c; D d; E e; B *p = &e, *q = &c;
  v.push_back(&c); v.push_back(&b); v.push_back(&d); v.push_back(&c);
  v.push_back(&e); v.push_back(p);
  cout << Fun(v); // stampa 2
  w.push_back(p); w.push_back(&d); w.push_back(q); w.push_back(&e);
  cout << Fun(w); // stampa 1
}
```

```
int Fun(const vector<B*>& v) {
  int tot = 0;
  for(auto it = v.begin(); it!= v.end();
      ++it)
    if(typeid(*v[0])!= typeid(*(*it)) &&
       dynamic_cast<C*>(*it)) ++tot;
  return tot;
}
```