```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

```
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() "; }
};
```

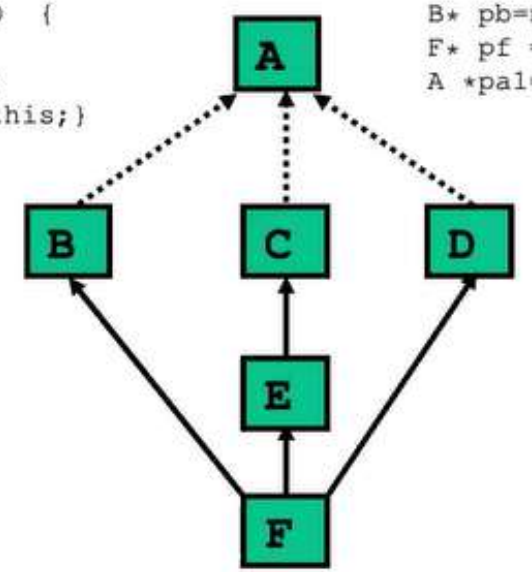
```
class B: virtual public A {
 public:
  void f(const bool&){cout<< "B::f(const bool&) ";}
  void f(const int&){cout<< "B::f(const int&) ";}
  virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
  virtual ~B() {cout << "~B ";}
  B() {cout <<"B() "; }
};
```

```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

```
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() "; }
};
```

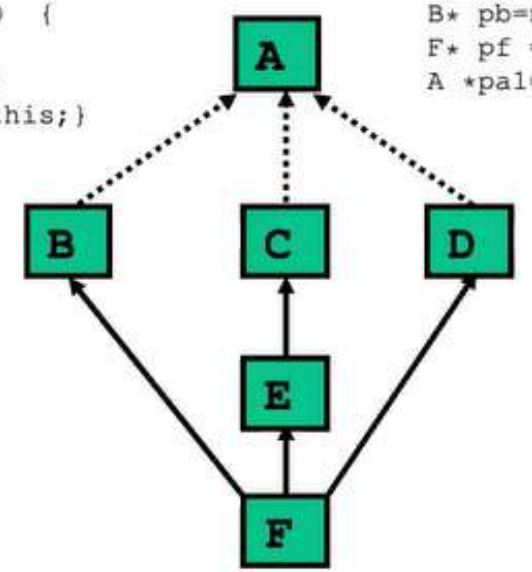
```
class B: virtual public A {
 public:
  void f(const bool&){cout<< "B::f(const bool&) ";}
  void f(const int&){cout<< "B::f(const int&) ";}
  virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
  virtual ~B() {cout << "~B ";}
  B() {cout <<"B() "; }
};
```

```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

```
pa3->f(3);

pa5->f(3);
```

```
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() "; }
};
```

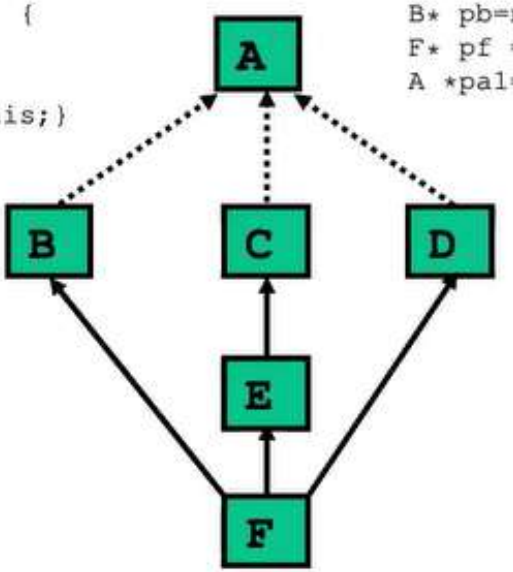
```
class B: virtual public A {
 public:
  void f(const bool&){cout<< "B::f(const bool&) ";}
  void f(const int&){cout<< "B::f(const int&) ";}
  virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
  virtual ~B() {cout << "~B ";}
  B() {cout <<"B() "; }
};
```

```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

```
pb1->f(true);

pa4->f(true);
```

```
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() "; }
};
```

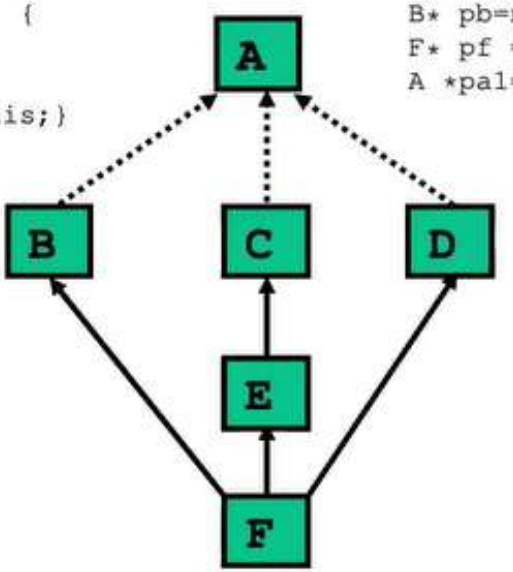
```
class C: virtual public A {
 public:
  C* f(Z){cout <<"C::f(Z) "; return this;}
  C() {cout <<"C() "; }
};
```

```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```




```
pa2->f(Z(2));

pa5->f(Z(2));
```

```
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() "; }
};
```

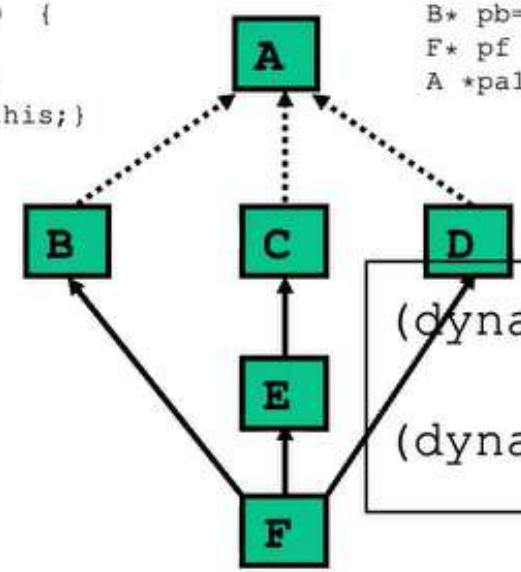
```
class C: virtual public A {
 public:
  C* f(Z){cout <<"C::f(Z) "; return this;}
  C() {cout <<"C() "; }
};
```

```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```




```
(dynamic_cast<E*>(pa4))->f(Z(2));

(dynamic_cast<C*>(pa5))->f(Z(2));
```

```
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() "; }
};
```

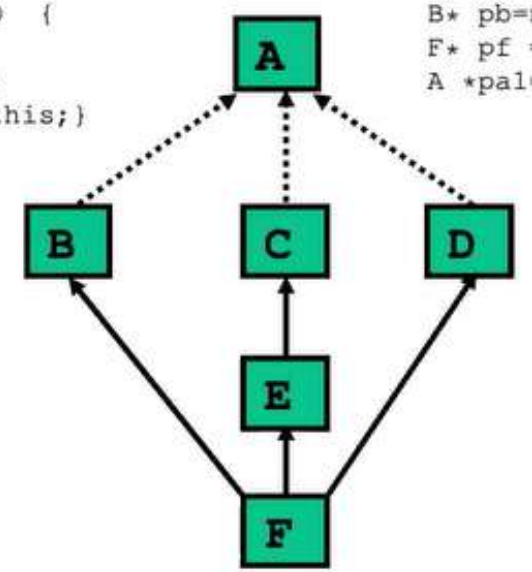
```
class B: virtual public A {
 public:
  void f(const bool&){cout<< "B::f(const bool&) ";}
  void f(const int&){cout<< "B::f(const int&) ";}
  virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
  virtual ~B() {cout << "~B ";}
  B() {cout <<"B() "; }
};
```

```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

```
pb->f(3);

pc->f(3);
```

```
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() "; }
};
```

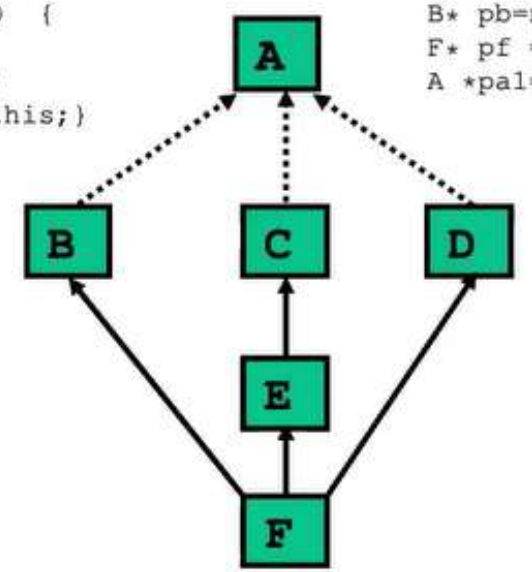
```
class C: virtual public A {
 public:
  C* f(Z){cout <<"C::f(Z) "; return this;}
  C() {cout <<"C() "; }
};
```

```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```




```
(pa4->f(Z(3)))->f(4);

(pc->f(Z(3)))->f(4);
```

```
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() "; }
};
```

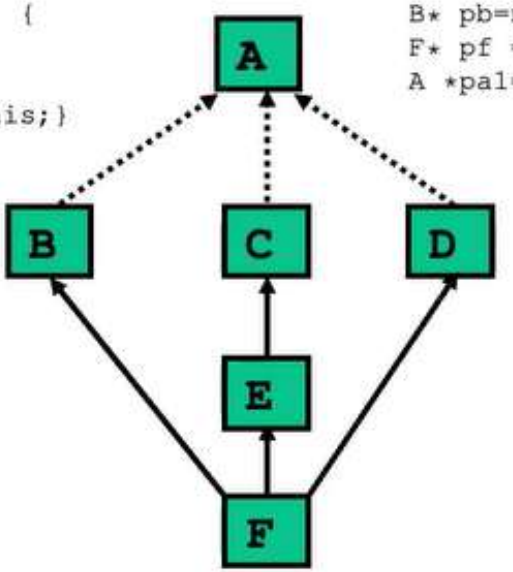
```
class B: virtual public A {
 public:
  void f(const bool&){cout<< "B::f(const bool&) ";}
  void f(const int&){cout<< "B::f(const int&) ";}
  virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
  virtual ~B() {cout << "~B ";}
  B() {cout <<"B() "; }
};
```

```
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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

```
E* puntE = new F;

delete pa5;

delete pb1;
```

cosa stampa?

```
pa3->f(3);
pa5->f(3);
pb1->f(true);
pa4->f(true);
pa2->f(Z(2));
pa5->f(Z(2));
(dynamic_cast<E*>(pa4))->f(Z(2));
(dynamic_cast<C*>(pa5))->f(Z(2));
pb->f(3);
pc->f(3);
(pa4->f(Z(3)))->f(4);
(pc->f(Z(3)))->f(4);
E* puntE = new F;
delete pa5;
delete pb1;
```

```
A::f(int) A::f(bool)
A::f(int) F::f(bool)
B::f(const bool&)
A::f(bool)
C::f(Z)
F::f(Z)
E::f(Z)
F::f(Z)
B::f(const int&)
C::f(Z)
E::f(Z) A::f(int) A::f(bool)
C::f(Z) C::f(Z)
A() B() C() E() D() F()
NESSUNA STAMPA
~F ~D ~E ~B
```

Si consideri la gerarchia di classi per l'I/O. La classe base `ios` ha il distruttore virtuale, il costruttore di copia privato ed un unico costruttore (a 2 parametri con valori di default) protetto. Diciamo che le classi derivate da `istream` ma non da `ostream` (ad esempio `ifstream`), e `istream` stessa, sono *classi di input*, le classi derivate da `ostream` ma non da `istream` (ad esempio `ofstream`), ed `ostream` stessa, sono *classi di output*, mentre le classi derivate sia da `istream` che da `ostream` sono *classi di I/O* (esempi: `iostream` e `fstream`). Quindi ogni classe di input, output o I/O è una sottoclasse di `ios`. Definire una funzione `int F(ios& ref)` che restituisce -1 se il tipo dinamico di `ref` è un riferimento ad una classe di input, 1 se il tipo dinamico di `ref` è un riferimento ad una classe di output, 0 se il tipo dinamico di `ref` è un riferimento ad una classe di I/O, mentre in tutti gli altri casi ritorna 9.

Quindi, ad esempio, il seguente `main()` provoca la stampa riportata.

```
class D : public ios {
};

main() {
  istream& b = cin;
  ostream& c = cout;
  stringstream d;
  ifstream e("pippo");
  ofstream f("pluto");
  D g;
  cout << F(b) << ' ' << F(c) << ' ' << F(d) << ' ' << F(e) << ' '
       << F(f) << ' ' << F(g) << endl;
  // stampa: -1 1 0 -1 1 9
}
```

```cpp
/*
Si consideri la gerarchia di classi per l'I/O.
La classe base ios ha il distruttore virtuale, il costruttore di copia privato ed un unico costruttore (a 2 parametri con valori di default) protetto.
Diciamo che le classi derivate da istream ma non da ostream (ad esempio ifstream), e istream stessa, sono classi di input "puro",
le classi derivate da ostream ma non da istream (ad esempio ofstream), ed ostream stessa, sono classi di output,
mentre le classi derivate sia da istream che da ostream sono classi di I/O (esempi: iostream e fstream).
Quindi ogni classe di input, output o I/O e` una sottoclasse di ios. Definire una funzione int F(ios& ref)che restituisce -1 se il tipo dinamico
di ref e`un riferimento ad una classe di input,1 se il tipodinamico di ref e` un riferimento ad una classe di output, 0 se il tipo dinamico di ref
e` un riferimento ad una classe di I/O, mentre in tutti gli altri casi ritorna 9.
*/


int F(std::ios& ref) {
  if(dynamic_cast<std::istream*>(&ref) && !(dynamic_cast<std::ostream*>(&ref))) return -1;
  if(!dynamic_cast<std::istream*>(&ref) && dynamic_cast<std::ostream*>(&ref)) return 1;
  if(dynamic_cast<std::istream*>(&ref) && dynamic_cast<std::ostream*>(&ref)) return 0;
  return 9;
}

```

Ognuno dei seguenti frammenti è il codice di uno o piú metodi pubblici di una qualche classe C. La loro compilazione provoca errori?

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

Ognuno dei seguenti frammenti è il codice di uno o piú metodi pubblici di una qualche classe C. La loro compilazione provoca errori?

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

```
class A {
public:
  virtual void m() =0;
};

class B: virtual public A {};

class C: virtual public A {
public:
  virtual void m() {}
};

class D: public B, public C {
public:
  virtual void m() {}
};

class E: public D {};

class F: public E {};

char G(A* p, B& r) {
  C* pc = dynamic_cast<E*>(&r);
  if(pc && typeid(*p)==typeid(r)) return 'G';
  if(!dynamic_cast<E*>(&r) && dynamic_cast<D*>(p)) return 'Z';
  if(!dynamic_cast<F*>(pc)) return 'A';
  else if(typeid(*p)==typeid(E)) return 'S';
  return 'E';
}
```

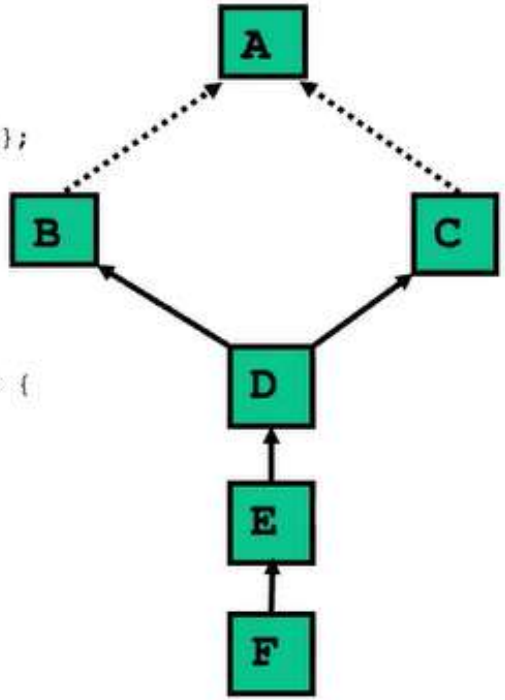
Si consideri inoltre il seguente statement.

```
  cout << G(new X1,*new Y1) << G(new X2,*new Y2) << G(new X3,*new Y3) << G(new X4,*new Y4)
       << G(new X5,*new Y5) << G(new X6,*new Y6) << G(new X7,*new Y7) << G(new X8,*new Y8);
```

Definire opportunamente le incognite di tipo `Xi` e `Yi` tra i tipi `A`, `B`, `C`, `D`, `E`, `F` della precedente gerarchia in modo tale che:

1. Lo statement non includa piú di una chiamata della funzione `G` con gli stessi parametri attuali
2. La compilazione dello statement non produca illegalità
3. L'esecuzione dello statement non provochi errori a run-time
4. L'esecuzione dello statement produca in output esattamente la stampa **SAGGEZZA**.

```
class A {
public:
  virtual void m() =0;
};

class B: virtual public A {};

class C: virtual public A {
public:
  virtual void m() {}
};

class D: public B, public C {
public:
  virtual void m() {}
};

class E: public D {};

class F: public E {};

char G(A* p, B& r) {
  C* pc = dynamic_cast<E*>(&r);
  if(pc && typeid(*p)==typeid(r)) return 'G';
  if(!dynamic_cast<E*>(&r) && dynamic_cast<D*>(p)) return 'Z';
  if(!dynamic_cast<F*>(pc)) return 'A';
  else if(typeid(*p)==typeid(E)) return 'S';
  return 'E';
}
```



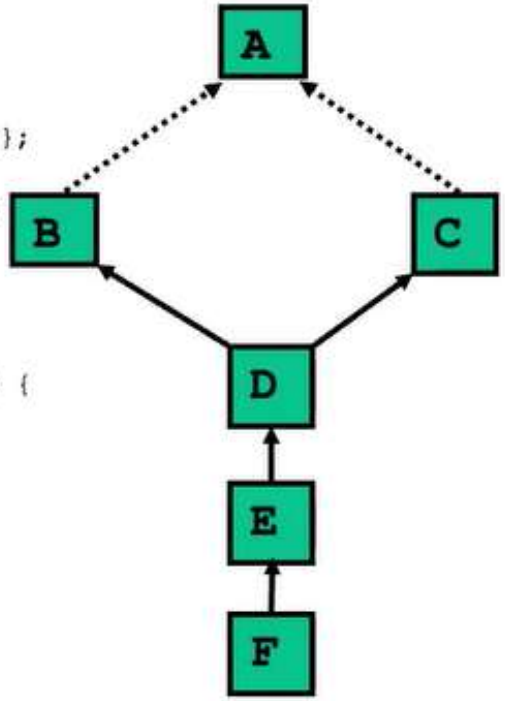
Si consideri inoltre il seguente statement.

```
  cout << G(new X1,*new Y1) << G(new X2,*new Y2) << G(new X3,*new Y3) << G(new X4,*new Y4)
       << G(new X5,*new Y5) << G(new X6,*new Y6) << G(new X7,*new Y7) << G(new X8,*new Y8);
```

Definire opportunamente le incognite di tipo `Xi` e `Yi` tra i tipi `A`, `B`, `C`, `D`, `E`, `F` della precedente gerarchia in modo tale che:

1. Lo statement non includa piú di una chiamata della funzione `G` con gli stessi parametri attuali
2. La compilazione dello statement non produca illegalità
3. L'esecuzione dello statement non provochi errori a run-time
4. L'esecuzione dello statement produca in output esattamente la stampa **`SAGGEZZA`**.

```
class A {
public:
  virtual void m() =0;
};

class B: virtual public A {};

class C: virtual public A {
public:
  virtual void m() {}
};

class D: public B, public C {
public:
  virtual void m() {}
};

class E: public D {};

class F: public E {};

char G(A* p, B& r) {
  C* pc = dynamic_cast<E*>(&r);
  if(pc && typeid(*p)==typeid(r)) return 'G';
  if(!dynamic_cast<E*>(&r) && dynamic_cast<D*>(p)) return 'Z';
  if(!dynamic_cast<F*>(pc)) return 'A';
  else if(typeid(*p)==typeid(E)) return 'S';
  return 'E';
}
```




```
TD(*p) ∈ {C,D,E,F}
TD(r) ∈ {D,E,F}

output G ∈ {(E,E), (F,F)}
TD(r) ≦ E & TD(*p) = TD(r)

output Z ∈ {(D,D), (E,D), (F,D)}
¬G & TD(r) ≦̸ E & TD(*p) ≦ D

output A ∈ {(C,D), (C,E) (D,E), (F,E)}
¬Z & ¬G & TD(r) ≦̸ F

output S ∈ {(E,F)}
¬Z & ¬G & ¬A & TD(r) = F & TD(*p)=E

output E ∈ {(C,F), (D,F)}
¬Z & ¬G & ¬A & ¬S
```

Si consideri inoltre il seguente statement.

```
  cout << G(new X1,*new Y1) << G(new X2,*new Y2) << G(new X3,*new Y3) << G(new X4,*new Y4)
       << G(new X5,*new Y5) << G(new X6,*new Y6) << G(new X7,*new Y7) << G(new X8,*new Y8);
```

Definire opportunamente le incognite di tipo `Xi` e `Yi` tra i tipi `A`, `B`, `C`, `D`, `E`, `F` della precedente gerarchia in modo tale che:

1. Lo statement non includa piú di una chiamata della funzione `G` con gli stessi parametri attuali
2. La compilazione dello statement non produca illegalità
3. L'esecuzione dello statement non provochi errori a run-time
4. L'esecuzione dello statement produca in output esattamente la stampa **SAGGEZZA**.