```
class B {
public:
  B() {cout << " B() ";}
                                                                      class C: virtual public B {
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                      public:
  virtual void g() const {cout <<" B::g ";}</pre>
                                                                        C() {cout<< " C() ";}
  virtual const B* i() {cout<<" B::j "; n(); return this;}</pre>
                                                                         ~C() {cout<< " ~C() ";}
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
                                                                        void g() const {cout <<"pre><"pre>cout <<"pre>cout
  void m() {cout <<" B::m_"; g(); j();}</pre>
                                                                        void k() override {cout >
                                                                                                        C::k "; B::n();}
  virtual B& n() {cout < "B::n "; return *this;}
                                                                        virtual void m() {cout << " C::m "; g(); j();}</pre>
};
                                                                        B& n() override {cout << " C::n "; return *this;}
class D: virtual public B
                                                                      class E: public C, public D {
public:
                                                                      public:
  D() {cout<< " D() ";}
                                                                        E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                         ~E() {cout<< " ~E() ";}
  virtual void g() {cout <<" D::g ";}</pre>
                                                                        virtual void g() const {coux
                                                                                                             E::q ";}
                                                                        const E* j() {cout <<" E:: "; return this;}
void m() {cout <<" E:: ""; g(); j();}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
  void k() const {cout <<" D::k "; k();}</pre>
  void m() {cout <<" D::m "; g(); j();}</pre>
                                                                        D& n() final {cout << " E::n "; return *this;}
                                                                      }:
}:
class F: virtual public E {
                                                                       CPUNDO .ml
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}</pre>
                                                              B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
  void m() {cout <<" F::m "; j();}</pre>
                                                              C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m(); .....
p2->m();
C* p = \text{new } F(F());
(dynamic_cast<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p5))->j(); .....
```

```
class B {
public:
  B() {cout << " B() ";}
                                                                         class C: virtual public B {
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                         public:
  virtual void g() const {cout <<" B::g ";}
                                                                           C() {cout<< " C() ";}
  virtual const B* j() {cout<<" B::j "; n(); return this;}</pre>
                                                                           ~C() {cout<< " ~C() ";}
  virtual void k() {cout <<" B:: "; j(); m(); }
void m() {cout <<" B::m "; g(); j();}</pre>
                                                                           void g() const {cout <<" C::g ";}</pre>
                                                                           void k() override {cout <<" C::k "; B::n();}</pre>
                            <" B::n ";
                                                                           virtual void m() {cout << " C::m "; g(); j();}</pre>
  virtual B& n() {cout~
                                         return *this; }
                                                                           B& n() override {cout << " C::n "; return *this;}
};
                                                                         };
class D: virtual public B
                                                                        class E: public C, public D {
public:
                                                                         public:
  D() {cout<< " D() ";}
                                                                           E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                           ~E() {cout<< " ~E() ";}
  virtual void g() {cout <<" D::g ";}</pre>
                                                                           virtual void g() const {cout <<" E::g ";}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
                                                                           const E* j() {cout <<" E::j "; return this;}</pre>
  void k() const {cout << " D::x ";
                                                                           void m() {cout <<" E::m "; g(); j();}</pre>
  void m() {cout <<" D::m "; g();</pre>
                                                                           D& n() final {cout << " E::n "; return *this;}
                                      i(
};
                                                                         1:
class F: virtual public E {
public:
                                                                                  P3->KL)-
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}</pre>
                                                                B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
  void m() {cout <<" F::m "; j();}</pre>
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();
p3->k();
(p3->n()).m();

p2->m();
(p2->j())->g();

C* p = new F(F());
(p1->j())->k();
(dynamic_cast<const F*>(p1->j()))->g();
(dynamic_cast<E*>(p5))->j();
(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```

```
class B {
public:
  B() {cout << " B() ";}
                                                                     class C: virtual public B {
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                     public:
  virtual void g() const {cout <<" B::g</pre>
                                                                       C() {cout<< " C() ";}
  virtual const B* j() {cout<<" B::j "; n(); return this;}</pre>
                                                                       ~C() {cout<< " ~C() ";}
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
                                                                       void g() const {cout <<" C::g ";}</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
                                                                       void k() override {cout <<" C::k "; B::n();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
                                                                       virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                       B& n() override {cout << " C::n "; return *this;}
};
                                                                     };
class D: virtual public B {
                                                                     class E: public C, public D {
public:
                                                                     public:
  D() {cout<< " D() ";}
                                                                       E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                       ~E() {cout<< " ~E() ";}
  virtual void g() {cout << " D':g ";}</pre>
                                                                       virtual void g() const {cout <<" E::g ";}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
                                                                       const E* j() {cout <<" E::j "; return this;}</pre>
  void k() const {cout << " D::k "; k();}</pre>
                                                                       void m() {cout <<" E::m "; g(); j();}</pre>
  void m() {cout <<" D::m "; g(); j();}</pre>
                                                                       D& n() final {cout << " E::n "; return *this;}
1:
                                                                     }:
class F: virtual public E {
                                                                       (R3 >NL)) .nl)
public:
  F() {cout<< " F() ";}
   ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}</pre>
                                                             B* p1 = new E(); B* p2 = new C(); B* p3
  void m() {cout <<" F::m "; j();}</pre>
                                                             C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();

p3->k();

(p3->n()).m();

p2->m();

(p2->j())->g();

C* p = new F(F());

(p1->j())->k();

(dynamic_cast<const F*>(p1->j()))->g();

(dynamic_cast<E*>(p5))->j();

(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```

P2 >7().

Esercizio Cosa Stampa

```
class B {
public:
 B() {cout << " B() ";}
                                                                    class C: virtual public B {
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                    public:
  virtual void g() const {cout << " #::g
                                                                      C() {cout<< " C() ";}
  virtual const B* j() {cout<<" B:/j
                                                                       ~C() {cout<< " ~C() ";}
                                        "; (n();
                                                return this; }
                                     "; j(); m(); }
  virtual void k() {cout <<" B: k
                                                                      void g() const {cout << " C::g ";}</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
                                                                      void k() override {cout << C::k "; B::n();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
                                                                      virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                      B& n() override {cout << " C::n "; return *this;}
};
class D: virtual public B {
                                                                    class E: public C, public D {
public:
                                                                    public:
 D() {cout<< " D() ";}
                                                                      E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                      ~E() {cout<< " ~E() ";}
 virtual void g() {cout <<" D::g ";}</pre>
                                                                      virtual void g() const {cout <<" E::g ";}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
                                                                      const E* j() {cout <<" E::j "; return this;}</pre>
  void k() const {cout <<" D::k "; k();}</pre>
                                                                      void m() {cout <<" E::m "; g(); j();}</pre>
 void m() {cout <<" D::m "; g(); j();}</pre>
                                                                      D& n() final {cout << " E::n "; return *this;}
1:
                                                                    }:
class F: virtual public E {
public:
 F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
 F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}</pre>
                                                            B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
                                                            C* p4 = new E(); const B* p5  new E(); const B* p6 = new F();
  void m() {cout <<" F::m "; j();}</pre>
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();
p3->k();
(p3->n()).m();

p2->m();
(p2->j())->g();

C* p = new F(F());
(p1->j())->k();
(dynamic_cast<const F*>(p1->j()))->g();
(dynamic_cast<E*>(p5))->j();
(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```

```
10 PJ
class B {
public:
                                                                   class C: virtual public B {
  B() {cout << " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                   public:
  virtual void g() const {cout <<" B::g ";}</pre>
                                                                      C() {cout<< " C() ";}
  virtual const B* j() {cout<<" B::j "; n(); return
virtual void k() {cout <<" B::k "; j(); m(); )</pre>
                                                                      ~C() {cout<< " ~C() ";}
                                                        this; }
                                                                      void g() const {cout <<" C::g ";}</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
                                                                      void k() override {cout <<" C::k "; B::n();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
                                                                      virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                      B& n() override {cout << " C::n "; return *this;}
};
class D: virtual public B {
                                                                   class E: public C, public D {
public:
                                                                   public:
  D() {cout<< " D() ";}
                                                                      E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                      ~E() {cout<< " ~E() ";}
  virtual void g() {cout <<" D::g ";}</pre>
                                                                      virtual void g() const {cout <<" E::g ";}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
                                                                      const E* j() {cout <<" E::j "; return this;}</pre>
  void k() const {cout <<" D::k "; k();}</pre>
                                                                      void m() {cout <<" E::m "; g(); j();}</pre>
  void m() {cout <<" D::m "; g(); j();}</pre>
                                                                     D& n() final {cout << " E::n "; return *this;}
1:
                                                                   1:
class F: virtual public E {
                                                                     P2-> > C)) -> G(); -> NC
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
                                                            void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" F::m "; j();}</pre>
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();
p3->k();
(p3->n()).m();
p2->m();
(p2->j())->g();
C* p = new F(F());
(p1->j())->k();
(dynamic_cast<const F*>(p1->j()))->g();
(dynamic_cast<E*>(p5))->j();
(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```

```
class B {
public:
  B() {cout << " B() ":}
                                                                        class C: virtual public B {
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                         public:
                                                                           C() {cout<< "_C() ";}
~C() {cout<< " ~C() ";}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; n(); return this;}
virtual void k() {cout <<" B::k "; j(); m(); }</pre>
                                                                           void g() const {cout << " C::g ";}</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
                                                                           void k() override {cout <<" C::k "; B::n();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
                                                                           virtual void m() {cout << " C::m "; g(); j();}</pre>
};
                                                                          B& n() override {cout << " C::n "; return *this;}
                                                                        };
                                                                        class E: public C, public D {
class D: virtual public B {
public:
                                                                         public:
                                                                           E() {cout<< " E() ";}
  D() {cout<< " D() ";}
                                                                           ~E() {cout<< "E() ";}
  ~D() {cout<< " D() ";}
  virtual void g() {cout <<" D::g ";}</pre>
                                                                           virtual void g() const {cout <<" E::g ";}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
                                                                           const E* j() {cout <<" E::j "; return this;}</pre>
  void k() const {cout <<" D::k "; k();}</pre>
                                                                           void m() {cout <<" E::m "; g(); j();}</pre>
  void m() {cout <<" D::m "; g(); j();}</pre>
                                                                           D& n() final {cout << " E::n "; return *this;}
};
                                                                         };
class F: virtual public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}</pre>
                                                                B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
  void m() {cout <<" F::m "; j();}</pre>
                                                                C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

Queste definizioni compilano correttamente (con opportuni #include e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();
p3->k();
(p3->n()).m();
p2->m();
(p2->j())->g();

C* p = new F(G();
(p1->j())->k();
(dynamic_cast<Const F*>(p1->j()))->g();
(dynamic_cast<E*>(p5))->j();
(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```

```
C* p=new F(F());

Questa istruzione esegue due costruzioni:

1. F() - crea un oggetto temporaneo F

2. new F(F()) - crea un nuovo oggetto F usando il copy constructor

Prima costruzione F():

• B() → C() → D() → E() → F()

Seconda costruzione new F(F()): Il copy constructor di F è definito come:

cpp

F(const F& x): B(x) {cout<<"Fc ";}
```

BASE
VILTURIO

BUSINE

```
class B {
public:
 B() {cout << " B() ";}
                                                                      class C: virtual public B {
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                      public:
  virtual void g() const {cout <<" B::g ";}</pre>
                                                                        C() {cout<< " C() ";}
 virtual const B* j() {cout<<" B::j "; n(); return this;}
virtual void k() {cout <<" B::k "; j(); m(); }</pre>
                                                                         ~C() {cout<< " ~C() ";}
                                                                        void g() const {cout <<" C::g ";}</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
                                                                        void k() override {cout <<" C::k "; B::n();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
                                                                        virtual void m() {cout << " C::m "; g(); j();}</pre>
};
                                                                        B& n() override {cout << " C::n "; return *this;}
class D: virtual public B {
                                                                      class E: public C, public D {
public:
                                                                      public:
                                                                                                                      4745
 D() {cout<< " D() ";}
                                                                        E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                        ~E() {cout<< " ~E() ";}
 virtual void g() {cout <<" D::g ";}</pre>
                                                                        virtual void g() const {cout <<" E::
 const B* j() {cout <<" D::j "; return this;}</pre>
                                                                        const E* j() {cout <<" E::j "; return this;}</pre>
  void k() const {cout <<" D::k "; k();}</pre>
                                                                        void m() {cout <<" E::m "; g(); j();}</pre>
 void m() {cout <<" D::m "; g(); j();}</pre>
                                                                        D& n() final {cout << " E::n "; return *this;}
}:
                                                                      }:
class F: virtual public E {
                                                                         (P1 > >0) >> KL).
public:
 F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
 F(const F& x): B(x) {cout<< " Fc ";}
 void k() {cout <<" F::k "; g();}</pre>
                                                              B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
 void m() {cout <<" F::m "; j();}</pre>
                                                                 p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();
p3->k();
(p3->n()).m();
p2->m();
(p2->j())->g();
C* p = new F(F());
(p1->j())->k();
(dynamic_cast<const F*>(p1->j()))->g();
(dynamic_cast<E*>(p5))->j();
(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```

CON ptis: >> PRITA CHIATO 5 POI

```
class B {
                                                   (SIA STATIC CLUS DYNANC)
public:
  B() {cout << " B() ";}
                                                                     class C: virtual public B {
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                     public:
  virtual void g() const {cout <<" B::g ";}</pre>
                                                                       C() {cout<< " C() ";}
 virtual const B* j() {cout<<" B::j "; n(); return this;}
virtual void k() {cout <<" B::k "; j(); m(); }</pre>
                                                                       ~C() {cout<< " ~C() ";}
                                                                       void g() const {cout <<" C::g ";}</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
                                                                       void k() override {cout <<" C::k "; B::n();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
                                                                       virtual void m() {cout << " C::m "; g(); j();}</pre>
};
                                                                       B& n() override {cout << " C::n "; return *this;}
                                                                     };
                                                                     class E: public C, public D {
class D: virtual public B {
public:
                                                                     public:
  D() {cout<< " D() ";}
                                                                       E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                       ~E() {cout<< " ~E() ";}
  virtual void g() {cout <<" D::g ";}</pre>
                                                                       virtual void g() const {cout <<" E::g ";}</pre>
                                                                       const E* j() {cout <<" E::j "; return this;}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
  void k() const {cout <<" D::k "; k();}</pre>
                                                                       void m() {cout <<" E::m "; g(); j();}</pre>
  void m() {cout <<" D::m "; g(); j();}</pre>
                                                                       D& n() final {cout << " E::n "; return *this;}
                                                       NC
};
                                                                     }:
class F: virtual public E {
public:
                                                              DIN-CASS (CONSFX) (P1->>()) ->6()
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}</pre>
                                                                p1 = new E(); B* p2 = new C(); B* p3 = new D();
  void m() {cout <<" F::m "; j();}</pre>
                                                             C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();
p3->k();
(p3->n()).m();
p2->m();
(p2->j())->g();
C* p = new F(F());
(p1->j())->k();
(dynamic_cast<const F*>(p1->j()))->g();
(dynamic_cast<E*>(p5))->j();
(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```

```
class B {
public:
  B() {cout << " B() ";}
                                                                      class C: virtual public B {
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                       public:
  virtual void g() const {cout <<" B::g ";}</pre>
                                                                         C() {cout<< " C() ";}
  virtual const B* j() {cout<<" B::j "; n(); return this;}
virtual void k() {cout <<" B::k "; j(); m(); }</pre>
                                                                         ~C() {cout<< " ~C() ";}
                                                                         void g() const {cout <<" C::g ";}</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
                                                                         void k() override {cout <<" C::k "; B::n();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
                                                                         virtual void m() {cout << " C::m "; g(); j();}</pre>
};
                                                                        B& n() override {cout << " C::n "; return *this;}
                                                                      class E: public C, public D {
class D: virtual public B {
public:
                                                                       public:
  D() {cout<< " D() ";}
                                                                         E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                         ~E() {cout<< " ~E() ";}
                                                                         virtual void g() const {cout <<" E::g ";}
  virtual void g() {cout <<" D::g ";}</pre>
                                                                         const E* j() {cout <<" E::j "; return this;}
void m() {cout <<" E::m "; g(); j();}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
  void k() const {cout <<" D::k "; k();}</pre>
  void m() {cout <<" D::m "; g(); j();}</pre>
                                                                         D& n() final {cout << " E::n "; return *this;}
1:
                                                                       }:
class F: virtual public E {
                                                             DTN-CAST (5 4>(PS) -> 50>.
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
                                                               B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
  void k() {cout <<" F::k "; q();}</pre>
  void m() {cout <<" F::m "; j();}</pre>
                                                               C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();
p3->k();
(p3->n()).m();
p2->m();
(p2->j())->g();
C* p = new F(F());
(p1->j())->k();
(dynamic_cast<const F*>(p1->j()))->g();
(dynamic_cast<E*>(p5))->j();
(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```

```
class B {
public:
                                                                     class C: virtual public B {
  B() {cout << " B() ":}
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                     public:
  virtual void g() const {cout <<" B::g ";}</pre>
                                                                       C() {cout<< " C() ";}
 virtual const B* j() {cout<<" B::j "; n(); return this;}
virtual void k() {cout <<" B::k "; j(); m(); }</pre>
                                                                       ~C() {cout<< " ~C() ";}
                                                                       void g() const {cout << " C::g</pre>
                                                                       void k() override {cout <<"(**)
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
                                                                       virtual void m() {cout << " C::m
                                                                                                          "; g(); j();}
};
                                                                       B& n() override {cout << " C::n "; return *this;}
                                                                   class E: public C, public D {
                                                                                                             C 4 PG = F()
class D: virtual public B {
public:
                                                                     public:
  D() {cout<< " D() ";}
                                                                       E() {cout<< " E() ";}
  ~D() {cout<< " ~D() ";}
                                                                       ~E() {cout<< " ~E() ";}
  virtual void g() {cout <<" D::g ";}</pre>
                                                                       virtual void g() const {cout <<" E::g ";}</pre>
                                                                       const E* j() {cout <<" E::j "; return this;}</pre>
  const B* j() {cout <<" D::j "; return this;}</pre>
  void k() const {cout <<" D::k "; k();}</pre>
                                                                       void m() {cout <<" E::m "; g(); j();}</pre>
  void m() {cout <<" D::m "; g(); j();}</pre>
                                                                       D& n() final {cout << " E::n "; return *this;
1:
                                                                     };
class F: virtual public E {
                                                            DIN_CAST LXXX (CONSI-CASI < B+7
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
                                                             B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
  void k() {cout <<" F::k "; q();}</pre>
  void m() {cout <<" F::m "; j();}</pre>
                                                             C* p4 = new E(); const B* p5 = new E(); const B* p6 = new
```

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori 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.

```
(p4->n()).m();
p3->k();
(p3->n()).m();

p2->m();
(p2->j())->g();

C* p = new F(F());
(p1->j())->k();
(dynamic_cast<const F*>(p1->j()))->g();
(dynamic_cast<E*>(p5))->j();
(dynamic_cast<C*>(const_cast<B*>(p6)))->k();
```