```
class B {
public:
  B() {cout << " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B∷g ";}
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k)) {cout <<" B::k "; j(); m(); }</pre>
                   <<" B::m "; g(); j();}
  void m() {cout
  virtual B& \n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                         class D: virtual public B {
public:
                                                                         public:
  C() {cout<< " C() ";}
                                                                           D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                            ~D() {cout<< " ~D() ";}
                                                                           virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                           const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                           void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                           void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                         class F: public E {
public:
                                                                         public:
  E() {cout<< " E() ";}
                                                                            F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                            ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}</pre>
                                                                           F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                           void k() {cout <<" F::k "; g();}</pre>
                                                                           void m() {cout <<" F::m "; j();}</pre>
  void m() {cout <<" E::m "; g(); <-j();}
  D& n() final {cout <<" E::n "; return *this;}
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E() const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F()
                                                                    new F(); F 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Ce>(p6)->j())->g();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(delete p7;
```

```
class B {
                                                                                    (e4 >NC). mC>.
public:
  B() {cout << "B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}
virtual B& n() {cout <>" B::n "; return *this;}
class C: virtual public B
                                                                      class D: virtual public B {
public:
                                                                      public:
  C() {cout<< " C() ";}
                                                                        D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                        ~D() {cout<< " ~D() ";}
                                                                        virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout << " C::g ";}</pre>
                                                                        const B* j() {cout <<" D::j "; return this;}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                        void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                        void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D
                                                                      class F: public E {
public:
                                                                      public:
  E() {cout<< " E() ";}
                                                                        F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                        ~F() {cout<< " ~F() ";}
  virtual void g() const
                                                                        F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout << E::
                                "; return this;}
                                                                        void k() {cout <<" F::k "; g();}</pre>
                                                                        void m() {cout <<" F::m "; j();}</pre>
  void m() {cout <<" E::m</pre>
                               g(); j();}
  D& n() final {cout <<"(::n) "; return *this;}
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 =
const B*p5 = new D(); const B*p6 = new E(); const B*p7 = new F(); F 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 behavioùr 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D&>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Cest F*>(p1->j())->g();
(dynamic_cast<Cest F*>(p1->j())->g();
(dynamic_cast<Cest F*>(p1->j())->k();
(dynamic_cast<Cest Center Cast<Cest Center Cast<Cest Center Cast C
```

```
class B {
public:
  B() {cout << "B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                                           P3->KO.
  virtual void f() {cout <<" B::f "</pre>
  virtual void g() const {cout << "B::g";}
  virtual const B* j() {cout<<" B
virtual void k() {cout << "B::}</pre>
                                    B::j "; return this;}
                                      "; j(); m(); }
  virtual void k() {cout << "(b:)
void m() {cout << "(b:) "; g()</pre>
                                    (i) j ()
                          <<" B::n '; return *this;}
  virtual B& n() {cout
class C: virtual public B {
                                                                         class D: virtual public B {
public:
                                                                         public:
  C() {cout<< " C() ";}
                                                                          D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                           ~D() {cout<< " ~D() ";}
  virtual void g() const override {cout <<" C::g ";}</pre>
                                                                          virtual void g() {cout <
                                                                          const B* j() {cout << ; ; ; return this;}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                          void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}
                                                                          void m() {cout <<" D::m "; g(); j();}</pre>
};
class E: public C, public D {
                                                                        class F: public E {
public:
                                                                        public:
  E() {cout<< " E() ";}
                                                                          F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                           ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                          F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                          void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                          void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = \text{new D()}; const B* p6 = \text{new E()}; const B* p7 = \text{new F()}; F 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Ce>(p6)->j())->g();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(delete p7;
```

```
class B {
public:
  B() {cout << "B() ";}
                                                                         ( ( 3 → N(), n(),
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f ";</pre>
  virtual void g() const {cout <<" (H::g
  virtual const B* j() {cout<<" B::j
                                           return this; }
  virtual void k() {cout <<" B::k "; j(
void m() {cout < (B::m "; g(); j();}</pre>
                           <<" B::k "; j(); m(); }
                                                              S
  virtual B& n() {cout <<"_B::n "; return *this;}</pre>
                                                                 9
class C: virtual public B {
                                                                      class D: virtual public B {
public:
                                                                      public:
  C() {cout<< " C() ";}
                                                                        D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                        ~D() {cout<< " ~D() ";}
                                                                        virtual void g() {cout << "D::g ";}
  virtual void g() const override {cout <<" C::g ";}</pre>
                                                                        const B* j() {cout <<"[L:j]"; return this;}
void k() const {cout <<"D::k "; k();}</pre>
  void k() override {cout <<" C::k "; B::n();}
virtual void m() {cout <<" C::m "; g(); j();}</pre>
  B& n() override {cout << " C::n "; return *this;}
                                                                        void m() {cout <<" D::m "; g(); j();}</pre>
};
class E: public C, public D {
                                                                      class F: public E {
public:
                                                                     public:
  E() {cout<< " E() ";}
                                                                        F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                        ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                        F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                        void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                       void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout << " E::n "; return *this;}
const B*p5 = new D(); const B*p6 = new E(); const B*p7 = new F(); F 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D&>(p3->n()).g();
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;
C* p = new F(f);
p1->m();
(p1->j())->k();
(dynamic_cast<Cost F*>(p1->j()))->g();
(dynamic_cast<Cost F*>(p6))->j();
(dynamic_cast<C>(const_cast<B>(p7)))->k();
delete p7;
```

```
(PYNAMIC CAST CDOE) (P3->NO)
class B {
public:
  B() {cout << "B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
class C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
  C() {cout<< " C() ";}
                                                                       D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                       ~D() {cout<< " ~D() ";}
                                                                       virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
                                                                       const B* j() {cout <<" D::j "; return this;}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                       void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                       void m() {cout <<" D::m "; g(); j();}</pre>
                                                                                                                            1
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                    public:
  E() {cout<< " E() ";}
                                                                       F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                       ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                       F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                      void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                      void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}
B* p1 = new E(); B* p2 = new C(); P* p3 = new D(); C* p4 = new E();
const B * p5 = \text{new D}(); const B * p6 = \text{new E}(); const B * p7 = \text{new F}(); F 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 behavioùr 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->f();

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

p3->k();

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

(dynamic_cast<D6>(p3->n())).g();

p2->f();

p2->m();

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

(p5->n()).g();

F x;

C* p = new F(f);

p1->m();

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

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

(dynamic_cast<Cest Cost_cast<B*>(p7)))->k();

delete p7;
```

```
class B {
public:
  B() {cout << " B() ";}
  virtual ~B() {cout<< " ~B() ";},</pre>
                                                                                   P2-2F(?
  virtual void f() {cout <<" B::f "; g(); j();}
virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j ": return this;</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                        class D: virtual public B {
public:
                                                                        public:
  C() {cout<< " C() ";}
                                                                          D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                          ~D() {cout<< " ~D() ";}
                                                                          virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout << " C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                          const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                          void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                          void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                        class F: public E {
public:
                                                                        public:
  E() {cout<< " E() ";}
                                                                          F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                          ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                          F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                          void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                          void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}</pre>
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B * p5 = \text{new D}(); const B * p6 = \text{new E}(); const B * p7 = \text{new F}(); F 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Ce>(p6)->j())->g();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(delete p7;
```

```
class B {
public:
  B() {cout << " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}
                                                                                P2->MU.
  virtual void g() const {cout <<" B::g</pre>
  virtual const B* j() {cout<<", :; return this;}
  virtual void k() {cout << B::k ; j(
void m() {cout << B::m ; g(); j();}</pre>

    B::k "; j(); m(); }

  virtual B& n() {cout <<
                             R::n "; return *this;}
class C: virtual public B
                                                                      class D: virtual public B {
public:
                                                                      public:
  C() {cout<< " C() ";}
                                                                        D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                        ~D() {cout<< " ~D() ";}
  virtual void g() const override {cout <<"(£::g
                                                                        virtual void g() {cout <<" D::g ";}</pre>
  void k() override {cout << " C::k "; B::n()
                                                                        const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                        void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}
                                                                        void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                      class F: public E {
public:
                                                                      public:
  E() {cout << " E() ";}
                                                                        F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                        ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                       F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                        void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                        void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}</pre>
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B * p5 = \text{new D()}; const B * p6 = \text{new E()}; const B * p7 = \text{new F()}; F 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;
C* p = new F(f);
p1->m();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<Cost F*>(p1->j())->k();
(dynamic_cast<Cost Cost_cast<B*>(p7)))->k();
delete p7;
```

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
                                                                      (p5\rightarrow n()).g();
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m()</pre>
                                                          NON COMPILA
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this
class C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
  C() {cout<< " C() ";}
                                                                       D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                       ~D() {cout<< " ~D() ";}
                                                                       virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                       const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                       void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                       void m() {cout <<" D::m "; g(); j();}</pre>
};
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                    public:
  E() {cout<< " E() ";}
                                                                       F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                       ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                      F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                      void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                      void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}
};
     new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();

p2->m();
(p2->j())->g();
(p5->n()).g();

F x;

C* p = new F(f);
p1->m();
(p1->j())->k();
(dynamic_cast<Cnst F*>(p1->j()))->g();
(dynamic_cast<Cr>
(dynamic_cast<Cr>
(dynamic_cast<Cr>
(const_cast<B*>(p7)))->k();
(dynamic_cast<Cr>
(dynamic_cast<Cr>
(const_cast<B*>(p7)))->k();
(delete p7;
```

```
class B {
public:
  B() {cout << " B() ";}
                                                                    C* p = new F(f);
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
  C() {cout<< " C() ";}
                                                                       D() {cout << " D()_";}
 C() {cout<< " ~C() ";}
                                                                       ~D() {cout<< " ~D() ";}
                                                                       virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                       const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                       void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                       void m() {cout <<" D::m "; g(); j();}</pre>
};
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                     public:
  E() {cout<< " E() ";}
                                                                       F() {cout << "F() ";}
                                                                       ~F() {cout<< " ~F() ";}
  ~E() {cout<< " ~E() ";}
                                                                      F(const F& x): B(x) {cout<< TFc
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                       void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                       void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B*p5 = new D(); const B*p6 = new E(); const B*p7 = new F(); F 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 behavioùr 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D&>(p3->n())).g();
p2->f();
p2->m();
(p5->n()).g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<Cost F*>(p1->j())->k();
(dynamic_cast<C*>(const_cast<B*>(p7)))->k();
delete p7;
```

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
                                                                        (p1\rightarrow j())\rightarrow k();
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                         class D: virtual public B {
public:
                                                                         public:
  C() {cout<< " C() ";}
                                                                           D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                           ~D() {cout<< " ~D() ";}
                                                                           virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                           const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                           void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                           void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                         class F: public E {
public:
                                                                         public:
  E() {cout << " E() ";}</pre>
                                                                           F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                            ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";</pre>
                                                                           F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E::j "; return
void m() {cout <<" E::m "; g(); j();}</pre>
                                                                           void k() {cout <<" F::k "; g();}</pre>
                                                                           void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}</pre>
      = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B*p5 = \text{new D}(); const B*p6 = \text{new E}(); const B*p7 = \text{new F}(); Ff;
```

- 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 behavioùr 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;
C* p = new F(f);
p1->m();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<Cost F*>(p1->j())->k();
(dynamic_cast<Cost Cost_cast<B*>(p7)))->k();
delete p7;
```

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                               (dynamic_cast < const F*>(p1 \rightarrow j())) \rightarrow g();
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}
virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
                                                                 SPRORE THIS CONSI
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                         class D: virtual public B
public:
                                                                         public:
  C() {cout<< " C() ";}
                                                                           D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                            ~D() {cout<< " ~D() ";}
                                                                           virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                           const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                           void k() const {cout <<" D::k "; k();}</pre>
                                                                           void m() {cout <<" D::m "; g(); j();}</pre>
  B& n() override {cout << " C::n "; return *this;}
};
                                                                    Class F: public E {
class E: public C, public D {
public:
                                                                         public:
  E() {cout << " E() ";}</pre>
                                                                           F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                            ~F() {cout<< " ~F() ";}
  virtual void g() const cout <<" E::g ";}
const E* j() {cout <<" E::i"; return this;}</pre>
                                                                           F(const F& x): B(x) {cout<< " Fc ";}
                                                                           void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}
                                                                           void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout << " E::n "; return *this;}
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B * p5 = \text{new D}(); const B * p6 = \text{new E}(); const B * p7 = \text{new F}(); F 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Ce>(p6)->j())->g();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(delete p7;
```

```
class B {
                                                                                                  CONST 5 X -> 5/
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
                                                                          (dynamic_cast < E > (p6)) \rightarrow j();
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
                                                                         class D: virtual public B {
class C: virtual public B {
public:
                                                                         public:
  C() {cout<< " C() ";}
                                                                           D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                           ~D() {cout<< " ~D() ";}
                                                                           virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout << " C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                           const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                           void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                           void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                         class F: public E {
public:
                                                                         public:
  E() {cout<< " E() ";}
                                                                           F() {cout << "F() ";}
                                                       BN
  ~E() {cout<< " ~E() ";}
                                                                            ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
const E* j() {cout <<" E::j "; return this;}</pre>
                                                                           F(const F& x): B(x) {cout<< " Fc ";}
                                                                           void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}
                                                                           void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout << " E::n "; return *this;}
                                                                         };
B* p1 = \text{new E}(); B* p2 = \text{new C}(); B* p3 = \text{new D}(); C* p4 = \text{new E}();
const B * p5 = \text{new D()}; const B * p6 = \text{new E()}; const B * p7 = \text{new F()}; F 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 behavioùr 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Ce>(p6)->j())->g();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(delete p7;
```

```
class B {
public:
  B() {cout << "B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
                                                            (dynamic_cast<C*>(const_cast<B*/(p7)))\rightarrow k();
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                       class D: virtual public B {
public:
                                                                       public:
  C() {cout<< " C() ";}
                                                                         D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                         ~D() {cout<< " ~D() ";}
                                                                         virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout <<" C::k "; B::n();}</pre>
                                                                         const B* j() {cout << " D::j "; return thi
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                         void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                         void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                       class F: public E {
public:
                                                                       public:
  E() {cout<< " E() ";}
                                                                         F() {cout << "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                         ~F() {cout<< " ~F() ";}
                                                                         F(const F& x): B(x) {cout << " Fc void k() {cout << " F::k"; g();}
  virtual void g() const {cout <<" 🛵 g
                {cout << " E::j "; return this;}
                                                                         void m()
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                                   {cout <<" F::m "; j();}
  D& n() final {cout << " E::n "; return *this;}
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = \text{new D()}; const B* p6 = \text{new E()}; const B* p7 = \text{new F()}; F 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D&>(p3->n())).g();
p2->f();
p2->m();
(p5->n()).g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<Cost F*>(p1->j())->k();
(dynamic_cast<C*>(const_cast<B*>(p7)))->k();
delete p7;
```

```
class B {
public:
                                                            VFN5NDNCNB
  B() {cout << "B() ";}
  virtual 'B() {cout<< " B() ";}
virtual void f() {cout << " B::f "; g();</pre>
  virtual void g() const {cout << " B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
                                                                                     delete p7;
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                          class D: virtual public B {
public:
                                                                          public:
  C() {cout<< " C() ";}
                                                                            D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                            ~D() {cout<< " ~D() ";}
                                                                            virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout << " C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                            const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                            void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}
                                                                            void m() {cout <<" D::m "; g(); j();}</pre>
                                                                          };
class E: public C, public D {
                                                                          class F: public E {
public:
                                                                         public:
  E() {cout << " E() ";}
                                                                            F() {cout<< "F() ";}
                                                                            F() {cout<< " ~F() ";}
F(const F& x): B(x) {cout<< " Fc ";}
   ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                            void k() {cout <<" F::k "; g();}</pre>
                                                                            void m() {cout <<" F::m "; j();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
  D& n() final {cout << " E::n "; return *this;}
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E(); const <math>B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F 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 behavioùr 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->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D&>(p3->n()).g();
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;
C* p = new F(f);
p1->m();
(p1->j())->k();
(dynamic_cast<Cost F*>(p1->j()))->g();
(dynamic_cast<Cost F*>(p6))->j();
(dynamic_cast<C>(const_cast<B>(p7)))->k();
delete p7;
```