```
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 D: virtual public B {
class C: virtual public B {
                                                                        public:
public:
                                                                         D() {cout<< "D()";}
~D() {cout<< "~D()";}
  C() {cout<< " C() ";}</pre>
  ~C() {cout<< " ~C() ";}
                                                                          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 F: public E {
class E: public C, public D {
public:
                                                                        public:
                                                                         F() {cout<< " F() ";}
F() {cout<< " F() ";}
  E() {cout<< " E() ";}
 ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}</pre>
                                                                         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(); 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**.

```
F x;
C* p = new F(f);
p1->f();
p1->m();
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); ......
p2->m();
(p2->i())->g():
p3->k(); .....
(p3->n()).m(); .....
(dynamic cast<D&>(p3->n())).g(); .....
p4->f();
p4->k();
(p4->n()).m();
(p5->n()).q();
(dynamic cast<E*>(p6))->j();
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); ......
delete p7; ......delete p7; .....
```

```
class B {
public:
 B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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 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 F: public E {
class E: public C, public D {
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
                                                                       void k() {cout <<" F::k "; g();}</pre>
  const E* j() {cout <<" E::j "; return this;}</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 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**.

```
F x;

C* p = new F(f);

pl->f();

pl->m();

(pl->j())->k();

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

p2->m();

(p2->j(),

p3->f();

p3->k();

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

(dynamic_cast<04>(p3->n())).g();

p4->f();

p4->k();

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

(dynamic_cast<8>(p6))->j();

(dynamic_cast<8>(const_cast<8)(p7)))->k();
```

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();
virtual void g() const {cout <<" B::g ";}</pre>
                                                                               ₽1 →¥ ().
  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 F: public E {
class E: public C, public D {
public:
                                                                          public:
  E() {cout<< " E() ";}
                                                                            F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                            ~F() {cout<< " ~F() ";}
  virtual void g() const {cout
                                                                            F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout << "E::j "; return this;}
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;}
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**.

```
F x;
C* p = new F(f);
pl->f();
p1->m();
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); .....
p2->m();
p3->k(); .....
(p3->n()).m(); .....
(dynamic cast<D&>(p3->n())).g(); .....
p4->k();
(p4->n()).m();
(p5->n()).g();
(dynamic cast<E*>(p6))->i();
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; ......
```

```
class B {
public:
                                                                                         p1 \rightarrow m()
  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;}</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 F: public E {
class E: public C, public D {
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 ";}
  const E* j() {cout <<" E::j"; return this;}
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;}
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}(); 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 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**.

```
F x;

C* p = new F(f);
pl->f();
pl->m();
(pl->j())->k();
(dynamic_cast<const F*>(pl->j()))->g();
p2->f();
p2->m();
(p2->j())->g();
p3->f();
p3->k();
(p3->n()).m();
(dynamic_cast<bb<(p3->n())).g();
p4->f();
p4->k();
(p4->n()).m();
(dynamic_cast<bb<(p3->n())).g();
p4->k();
(p5->n()).g();
(dynamic_cast<bb<(p3->n())).s();
(dynamic_cast<bb<(p3->n())).s();
(dynamic_cast<bb<(p3->n())).s();
```

```
class B {
public:
                                                                                 (p1→j())→k() >> NC
  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 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;}
                                                                        void m() {cout <<" D::m "; g(); j();}</pre>
                                                                      class F: public E {
class E: public C, public D {
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 <<" 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.

```
F x;

C* p = new F(f);
pl->f();
pl->m();
(pl->j())->k();
(dynamic_cast<const F*>(pl->j()))->g();
p2->f();
p2->m();
(p2->j())->g();
p3->k();
(p3->n()).m();
(dynamic_cast<Cb>(p3->n())).g();
p4->k();
(p4->n()).m();
((p5->n()).m();
((p5->n()).m();
((dynamic_cast<Cb>(p5)->j());
((dynamic_cast<Cb>(p5)->j();
(dynamic_cast<Cb>(const_cast<B>(p7)))->k();
(dynamic_cast<Cc>(const_cast<B>(p7)))->k();
(delete p7;
```

```
class B {
public:
  B() {cout<< " B() ";}
                                                                (dynamic_cast < const F*>((1\rightarrow j()))
  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 F: public E {
class E: public C, public D {
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 ";}
  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 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**.

```
F x; .....
C* p = new F(f);
p1->f();
p1->m(); .....
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); .....
p2->m(); ....
(p2->i())->g():
p3->k(); .....
(p3->n()).m(); .....
(dynamic cast<D&>(p3->n())).q(); .....
p4->f();
p4->k();
(p4->n()).m();
(p5->n()).g(); .....
(\text{dynamic cast} < E_* > (p6)) = > i():
(\text{dynamic cast} < C *> (\text{const cast} < B *> (p7))) -> k();
delete p7; ......
```

```
class B {
public:
  B() {cout<< " B() ";}
                                                                            (p2\rightarrow j())\rightarrow g()
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();
 virtual void g() const {cout << " B::g ", }
virtual const B* j() {cout << " B::j "; return this;}
virtual void k() {cout << " B::k "; j(); m(); }</pre>
                                                                                      P2 - > B/A
  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() const override {cout << "C::g</pre>
                                                                           virtual void g() {cout <<" D::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 F: public E {
class E: public C, public D {
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 ";}
  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 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**.

```
F x; .....
C* p = new F(f);
p1->f();
p1->m();
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); ......
p2->m();
p3->k(); .....
(p3->n()).m();
(dynamic cast<D&>(p3->n())).g(); .....
p4->f();
p4->k();
(p4->n()).m();
(p5->n()).q();
(dynamic_cast<E*>(p6))->j(); .....
(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>
  virtual void g() const {cout <<" B::g ";}</pre>
                                                                  (dynamic_cast<D&>(p3\rightarrow n())).g();
  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:
  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; }
                                                                        void k() const {cout <<" D::k "; k();}</pre>
  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 F: public E {
class E: public C, public D {
public:
                                                                     public:
                                                                       F() {cout<< " F() ";}
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
                                                                        ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}</pre>
                                                                       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(); 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**.

```
F x;

C* p = new F(f);

pl->f();

pl->m();

(pl->j())->k();

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

p2->f();

p2->m();

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

p3->k();

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

p4->f();

p4->k();

(p6->n()).m();

(dynamic_cast<E*>(p6))->j();

(dynamic_cast<C*>(const_cast<E*>(p7)))->k();

(dynamic_cast<C*>(const_cast<C*>(const_cast<E*>(p7)))->k();

(dynamic_cast<C*>(const_cast<E*>(p7)))->k();

(delete p7;
```

```
class B {
                                     "error: passing 'const B' as 'this' argument discards qualifiers"
public:
  B() {cout<< " B() ";}
                                                                             (p5\rightarrow n()).g();
  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;}
class C: virtual public B {
                                                                      class D: virtual public B {
                                                                      public:
public:
  C() {cout<< " C() ";}</pre>
                                                                        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 F: public E {
class E: public C, public D {
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 ";}
  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 \times p5 = \text{new D()}; const B \times p6 = \text{new E()}; const B \times 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**.

```
F x;
C* p = new F(f);
p1->f();
p1->m();
(p1->j())->k(); .....
(dynamic_cast<const F*>(p1->j()))->g(); ......
p2->m();
(p2->i())->g():
p3->k(); .....
(p3->n()).m(); .....
(dynamic cast<D&>(p3->n())).q();
p4->f();
p4->k();
(p4->n()).m();
(p5->n()).q();
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); ......
delete p7; ......
```

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                            (dynamic_cast<E*>(p6))\rightarrow j(); cout<<endl;
  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>
                                              cannot dynamic_cast 'p6' (of type 'const class B*') to type 'class E*'
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 F: public E {
class E: public C, public D {
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();}</pre>
           (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 = 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**.

```
F x;

C* p = new F(f);
pl->f();
pl->m();
(dynamic_cast<const F*>(pl->j()))>g();
p2->f();
p2->m();
(p2->j())->g();
p3->f();
p3->k();
(p3->n()).m();
(dynamic_cast<Cb>(p3->n())).g();
p4->f();
(p4->n()).m();
(g5->n()).m();
(g5->n()).m();
(dynamic_cast<B>(p3)->h()).m();
(dynamic_cast<B>(p3)->h();
(p4->n()).m();
(p4->n()).m();
(p5->n()).g();
(dynamic_cast<B>(p3)->h()).h();
```

```
UF NB ND NC NB
public:
  B() {cout<< " B() ";}
virtual ~B() {cout<< " ~B()</pre>
  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() ";}</pre>
                                                                       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 F: public E {
class E: public C, public D {
public:
                                                                     public:
  E() {cout<< " E() ";}
                                                                       F() {cout<< "F() ";}
                                                                     7 F() {cout<< " ~F() ";}
  ~E() {cout<< " ~E() ";}
  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 <<" 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**.

```
F x;

C* p = new F(f);

pl->f();

pl->m();

(pl->j())->k();

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

p2->f();

p2->m();

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

p3->k();

(dynamic_cast<D4>(p3->n())).m();

(dynamic_cast<D4>(p3->n())).g();

p4->k();

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

(dynamic_cast<E*>(p6))->j();

(dynamic_cast<C*>(const_cast<B*>(p7)))->k();

delete p7;
```