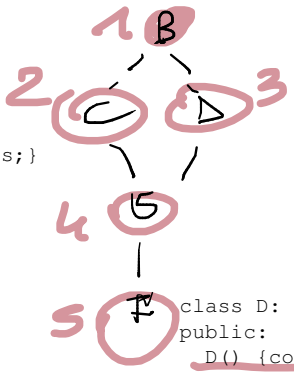
Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};

class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};

class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};
```



B-C-D-E-F

```
class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};

class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};

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;
```

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**.

```
F x; ..........

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

p1->f(); ..........

p1->m(); ..........

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

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

p2->f(); ..........

p2->m(); ..........

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

p3->f(); ..........

p3->k(); ..........

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

(dynamic_cast<D&>(p3->n())).g(); ..........

p4->f(); ..........

p4->k(); ..........

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

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

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

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

delete p7; ..........
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};

class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};

class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};

class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};

class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};

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;
```

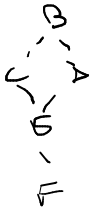
Queste definizioni compilano correttamente (con opportuni `#include` e `using`). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **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**.

```
F x; ..........

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

p1->f(); ..........

p1->m(); ..........

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

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

p2->f(); ..........

p2->m(); ..........

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

p3->f(); ..........

p3->k(); ..........

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

(dynamic_cast<D&>(p3->n())).g(); ..........

p4->f(); ..........

p4->k(); ..........

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

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

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

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

delete p7; ..........
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};
```



```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};
```

```
class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};
```

```
class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};
```

```
class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};
```

```
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;
```

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**.

```
F x; .....

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

p1->f(); .....

p1->m(); .....

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

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

p2->f(); .....

p2->m(); .....

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

p3->f(); .....

p3->k(); .....

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

(dynamic_cast<D&>(p3->n())).g(); .....

p4->f(); .....

p4->k(); .....

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

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

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

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

delete p7; .....
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};
```

p1→m()

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};
```

```
class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};
```

```
class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};
```

```
class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};
```

```
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;
```

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**.

```
F x; ..........

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

p1->f(); ..........

p1->m(); ..........

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

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

p2->f(); ..........

p2->m(); ..........

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

p3->f(); ..........

p3->k(); ..........

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

(dynamic_cast<D&>(p3->n())).g(); ..........

p4->f(); ..........

p4->k(); ..........

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

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

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

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

delete p7; ..........
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};

class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};

class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};

class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};

class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};

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;
```

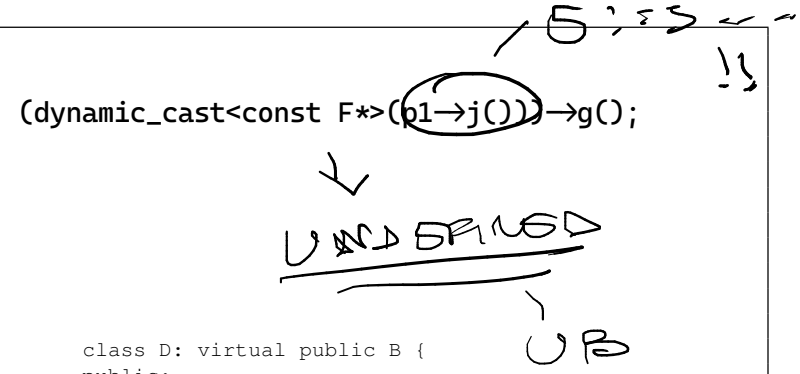
Queste definizioni compilano correttamente (con opportuni `#include` e `using`). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- **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**.

```
F x; ..........

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

p1->f(); ..........

p1->m(); ..........

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

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

p2->f(); ..........

p2->m(); ..........

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

p3->f(); ..........

p3->k(); ..........

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

(dynamic_cast<D&>(p3->n())).g(); ..........

p4->f(); ..........

p4->k(); ..........

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

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

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

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

delete p7; ..........
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};

class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};

class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};

class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};

class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};

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;
```



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**.

```
F x; .....

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

p1->f(); .....

p1->m(); .....

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

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

p2->f(); .....

p2->m(); .....

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

p3->f(); .....

p3->k(); .....

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

(dynamic_cast<D&>(p3->n())).g(); .....

p4->f(); .....

p4->k(); .....

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

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

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

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

delete p7; .....
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};
```

```
class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};
```

```
class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};
```

```
class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};
```

```
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;
```

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**.

```
F x; .....

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

p1->f(); .....

p1->m(); .....

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

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

p2->f(); .....

p2->m(); .....

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

p3->f(); .....

p3->k(); .....

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

(dynamic_cast<D&>(p3->n())).g(); .....

p4->f(); .....

p4->k(); .....

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

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

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

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

delete p7; .....
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};

class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};

class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};
```

```
(dynamic_cast<D&>(p3->n())).g();
```

```
class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};

class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};

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;
```

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**.

```
F x; .....

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

p1->f(); .....

p1->m(); .....

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

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

p2->f(); .....

p2->m(); .....

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

p3->f(); .....

p3->k(); .....

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

(dynamic_cast<D&>(p3->n())).g(); .....

p4->f(); .....

p4->k(); .....

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

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

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

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

delete p7; .....
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};
```

```
class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};
```

```
class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};
```

```
class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};
```

```
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;
```

"error: passing ‘const B’ as ‘this’ argument discards qualifiers"

(p5→n()).g();

NC

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**.

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

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};
```

```
class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};
```

```
class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};
```

```
class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};
```

```
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;
```

```
(dynamic_cast<E*>(p6))->j();cout<<endl;
```

```
cannot dynamic_cast 'p6' (of type 'const class B*') to type 'class E*'"
```

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**.

```
F x; ..........

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

p1->f(); ..........

p1->m(); ..........

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

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

p2->f(); ..........

p2->m(); ..........

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

p3->f(); ..........

p3->k(); ..........

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

(dynamic_cast<D&>(p3->n())).g(); ..........

p4->f(); ..........

p4->k(); ..........

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

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

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

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

delete p7; ..........
```

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  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(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};

class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};

class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};

class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};

class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};

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;
```

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**.

```
F x; .....

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

p1->f(); .....

p1->m(); .....

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

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

p2->f(); .....

p2->m(); .....

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

p3->f(); .....

p3->k(); .....

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

(dynamic_cast<D&>(p3->n())).g(); .....

p4->f(); .....

p4->k(); .....

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

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

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

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

delete p7; .....
```