

## Esercizio Cosa Stampa

```
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* 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;
```



PH → F C;

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* 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;
```



$(C \rightarrow N(C)), N(C),$

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))>g(); .....
(dynamic_cast<E*>(p6))>j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* 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;
```



P3 → k().

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))>g(); .....
(dynamic_cast<E*>(p6))>j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* 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;
```



(F3 → N(7), n(7),

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))>g(); .....
(dynamic_cast<E*>(p6))>j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* p1 = new E(); B* p2 = new C(); D* 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 COE) (P3 → N())

P3 → B / D

DYN → P3 → D / D

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* 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;
```

P2 → EC?



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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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;}
};
```

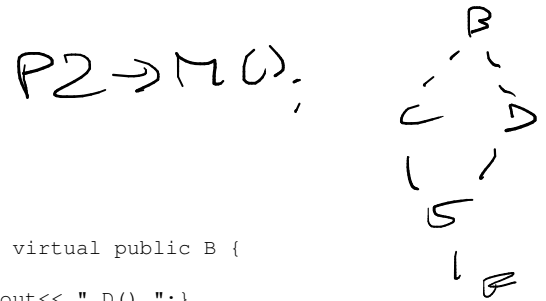
```
class C: virtual public B {
public:
    C() {cout<< " C() ";}
    ~C() {cout<< " ~C() ";}
    virtual void g() const override {cout <<" C::g ";}
    void k() override {cout <<" C::k "; B::n();}
    virtual void m() {cout <<" C::m "; g(); j();}
    B& n() override {cout <<" C::n "; return *this;}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

```
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* 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;
```

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



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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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;}
};
```

(p5→n()).g();

NON COMPILA

```
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**.

```
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<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```



## Esercizio Cosa Stampa

```
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;}
};
```

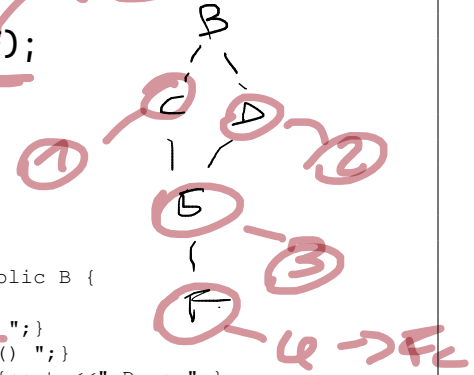
```
class E: public C, public D {
public:
    E() {cout<< " E() ";}
    ~E() {cout<< " ~E() ";}
    virtual void g() const {cout <<" E::g ";}
    const E* j() {cout <<" E::j "; return this;}
    void m() {cout <<" E::m "; g(); j();}
    D& n() final {cout <<" E::n "; return *this;}
};
```

```
B* 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;
```

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



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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))>g(); .....
(dynamic_cast<E*>(p6))>j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* 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;
```

(p1→j())→k();

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))>g(); .....
(dynamic_cast<E*>(p6))>j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

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

ERRORS THIS CONST  
PUNTA A NULL

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* 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();

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* 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;
```

~~const B\* p7~~

⊙ & p7 = new F()

(dynamic\_cast<C\*>(const\_cast<B\*>(p7)))→k();

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

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))->g(); .....
(dynamic_cast<E*>(p6))->j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); .....
delete p7; .....
```

## Esercizio Cosa Stampa

```
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* 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;
```

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

delete p7;

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

- **NON COMPILA** se la compilazione dello statement provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->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<const F*>(p1->j()))>g(); .....
(dynamic_cast<E*>(p6))>j(); .....
(dynamic_cast<C*>(const_cast<B*>(p7)))>k(); .....
delete p7; .....
```