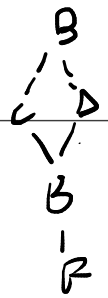
Esercizio Cosa Stampa



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

class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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 E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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

Esercizio Cosa Stampa

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

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

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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;}
};
```

P3->k();

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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

Esercizio Cosa Stampa

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

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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 E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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

Esercizio Cosa Stampa

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

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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 E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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

Esercizio Cosa Stampa

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

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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;}
};
```

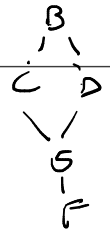
```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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

Esercizio Cosa Stampa




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

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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 E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

C* p = new F(F()); .....  NON SOLO OGNI OGGETTO SI COPIA

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

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

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

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

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

Questa istruzione esegue due costruzioni:

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

Prima costruzione F():

- B() → C() → D() → E() → F()

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

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

VIRTUALE

A()

B() C()

D()

BASE

VIRTUALE → B()

C() D() B() Fc()

Esercizio Cosa Stampa

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  virtual void g() const {cout <<" B::g ";}
  virtual const B* j() {cout<<" B::j "; n(); 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() ";}
  void g() const {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: virtual 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 E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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

Esercizio Cosa Stampa

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

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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 E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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

Esercizio Cosa Stampa

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

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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;}
};
```

DYN_CAST <B*>(p5) -> 5C>.

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D();
C* p4 = new E(); const B* p5 = new E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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

Esercizio Cosa Stampa

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

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  void g() const {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 E(); const B* p6 = new F();
```

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

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

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

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

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

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

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

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

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

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

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

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