

### Esercizio Cosa Stampa

```

class Z {
public: Z(int x) {}
};

class B: virtual public A {
public:
    void f(const bool&){cout<< "B::f(const bool&) ";}
    void f(const int&){cout<< "B::f(const int&) ";}
    virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
    virtual ~B() {cout << "~B ";}
    B() {cout <<"B() "; }
};

class D: virtual public A {
public:
    virtual void f(bool) const {cout <<"D::f(bool) ";}
    A* f(Z) {cout << "D::f(Z) "; return this;}
    ~D() {cout <<"~D ";}
    D() {cout <<"D() ";}
};

class F: public B, public E, public D {
public:
    void f(bool){cout << "F::f(bool) ";}
};

class A {
public:
    void f(int) {cout << "A::f(int) "; f(true);}
    virtual void f(bool) {cout <<"A::f(bool) ";}
    virtual A* f(Z) {cout <<"A::f(Z) "; f(2); return this;}
    A() {cout <<"A() "; }
};

class C: virtual public A {
public:
    C* f(Z){cout <<"C::f(Z) "; return this;}
    C() {cout <<"C() "; }
};

class E: public C {
public:
    C* f(Z){cout <<"E::f(Z) "; return this;}
    ~E() {cout <<"~E ";}
    E() {cout <<"E() ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pbl= new F;
A *pal=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
    
```

**Handwritten notes:**

- $PA3 \rightarrow F(3)$
- $A::f(int)$
- $A::f(bool)$
- $CONST! = NON CONST...$
- $A / D$

### Esercizio Cosa Stampa

```

class Z {
public: Z(int x) {}
};

class B: virtual public A {
public:
    void f(const bool&){cout<< "B::f(const bool&) ";}
    void f(const int&){cout<< "B::f(const int&) ";}
    virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
    virtual ~B() {cout << "~B ";}
    B() {cout <<"B() "; }
};

class D: virtual public A {
public:
    virtual void f(bool) const {cout <<"D::f(bool) ";}
    A* f(Z) {cout << "D::f(Z) "; return this;}
    ~D() {cout <<"~D ";}
    D() {cout <<"D() ";}
};

class F: public B, public E, public D {
public:
    void f(bool){cout << "F::f(bool) ";}
    F* f(Z){cout <<"F::f(Z) "; return this;}
    F() {cout <<"F() "; }
    ~F() {cout <<"~F ";}
};

class A {
public:
    void f(int) {cout << "A::f(int) "; f(true);}
    virtual void f(bool) {cout <<"A::f(bool) ";}
    virtual A* f(Z) {cout <<"A::f(Z) "; f(2); return this;}
    A() {cout <<"A() "; }
};

class C: virtual public A {
public:
    C* f(Z){cout <<"C::f(Z) "; return this;}
    C() {cout <<"C() "; }
};

class E: public C {
public:
    C* f(Z){cout <<"E::f(Z) "; return this;}
    ~E() {cout <<"~E ";}
    E() {cout <<"E() ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pbl= new F;
A *pal=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
    
```

**Handwritten notes:**

- $TS: A \rightarrow D \rightarrow F$
- $PA5 \rightarrow F(3)$
- $A::f(int)$
- $F::f(bool)$

### Esercizio Cosa Stampa

```

class Z {
public: Z(int x) {}
};

class B: virtual public A {
public:
    void f(const bool&){cout<< "B::f(const bool&) ";}
    void f(const int&){cout<< "B::f(const int&) ";}
    virtual B* f(Z) {cout <<"B::f(Z) "; return this;}
    virtual ~B() {cout << "~B ";}
    B() {cout <<"B() "; }
};

class D: virtual public A {
public:
    virtual void f(bool) const {cout <<"D::f(bool) ";}
    A* f(Z) {cout << "D::f(Z) "; return this;}
    ~D() {cout <<"~D ";}
    D() {cout <<"D() ";}
};

class F: public B, public E, public D {
public:
    void f(bool){cout << "F::f(bool) ";}
    F* f(Z){cout <<"F::f(Z) "; return this;}
    F() {cout <<"F() "; }
    ~F() {cout <<"~F ";}
};

class A {
public:
    void f(int) {cout << "A::f(int) "; f(true);}
    virtual void f(bool) {cout <<"A::f(bool) ";}
    virtual A* f(Z) {cout <<"A::f(Z) "; f(2); return this;}
    A() {cout <<"A() "; }
};

class C: virtual public A {
public:
    C* f(Z){cout <<"C::f(Z) "; return this;}
    C() {cout <<"C() "; }
};

class E: public C {
public:
    C* f(Z){cout <<"E::f(Z) "; return this;}
    ~E() {cout <<"~E ";}
    E() {cout <<"E() ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pbl= new F;
A *pal=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
    
```

**Handwritten notes:**

- $(B/F)$
- $PB1 \rightarrow F(true)$

## Esercizio Cosa Stampa

```
class Z {
public: Z(int x) {}
};
```

Z(2)

(dynamic\_cast<E\*>(pa4))->f(Z(2));

DA: A ← PA4 = NSWB

AS ← PA4 = NSWB

```
class B: virtual public A {
public:
    void f(const bool&) {cout << "B::f(const bool&) ";}
    void f(const int&) {cout << "B::f(const int&) ";}
    virtual B* f(Z) {cout << "B::f(Z) "; return this;}
    virtual ~B() {cout << "~B ";}
    B() {cout << "B() ";}
```

```
};

class D: virtual public A {
public:
    virtual void f(bool) const {cout << "D::f(bool) ";}
    A* f(Z) {cout << "D::f(Z) "; return this;}
    ~D() {cout << "~D ";}
    D() {cout << "D() ";}
```

```
};

class F: public B, public E, public D {
public:
    void f(bool) {cout << "F::f(bool) ";}
    F* f(Z) {cout << "F::f(Z) "; return this;}
    F() {cout << "F() ";}
    ~F() {cout << "~F ";}
```

```
};
```

```
class A {
public:
    void f(int) {cout << "A::f(int) "; f(true);}
    virtual void f(bool) {cout << "A::f(bool) ";}
    virtual A* f(Z) {cout << "A::f(Z) "; f(2); return this;}
    A() {cout << "A() ";}
```

B::f(Z) / A::f(int) / A::f(bool)

```
class C: virtual public A {
public:
    C* f(Z) {cout << "C::f(Z) "; return this;}
    C() {cout << "C() ";}
```

```
class E: public C {
public:
    C* f(Z) {cout << "E::f(Z) "; return this;}
    ~E() {cout << "~E ";}
    E() {cout << "E() ";}
```

```
B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B* pbl= new F;
A* pal=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

## Esercizio Cosa Stampa

```
class Z {
public: Z(int x) {}
};
```

B ← PA5 = NSWF

```
class B: virtual public A {
public:
    void f(const bool&) {cout << "B::f(const bool&) ";}
    void f(const int&) {cout << "B::f(const int&) ";}
    virtual B* f(Z) {cout << "B::f(Z) "; return this;}
    virtual ~B() {cout << "~B ";}
    B() {cout << "B() ";}
```

```
};

class D: virtual public A {
public:
    virtual void f(bool) const {cout << "D::f(bool) ";}
    A* f(Z) {cout << "D::f(Z) "; return this;}
    ~D() {cout << "~D ";}
    D() {cout << "D() ";}
```

```
};

class F: public B, public E, public D {
public:
    void f(bool) {cout << "F::f(bool) ";}
    F* f(Z) {cout << "F::f(Z) "; return this;}
    F() {cout << "F() ";}
    ~F() {cout << "~F ";}
```

```
};
```

```
class A {
public:
    void f(int) {cout << "A::f(int) "; f(true);}
    virtual void f(bool) {cout << "A::f(bool) ";}
    virtual A* f(Z) {cout << "A::f(Z) "; f(2); return this;}
    A() {cout << "A() ";}
```

```
class C: virtual public A {
public:
    C* f(Z) {cout << "C::f(Z) "; return this;}
    C() {cout << "C() ";}
```

```
class E: public C {
public:
    C* f(Z) {cout << "E::f(Z) "; return this;}
    ~E() {cout << "~E ";}
    E() {cout << "E() ";}
```

```
B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B* pbl= new F;
A* pal=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

```
class Z {
public: Z(int x) {}
};
```

DELETE PA5.  
(NOSUBS, PAMPA) ~ AC;

```
class B: virtual public A {
public:
    void f(const bool&) {cout << "B::f(const bool&) ";}
    void f(const int&) {cout << "B::f(const int&) ";}
    virtual B* f(Z) {cout << "B::f(Z) "; return this;}
    virtual ~B() {cout << "~B ";}
    B() {cout << "B() ";}
```

```
};

class D: virtual public A {
public:
    virtual void f(bool) const {cout << "D::f(bool) ";}
    A* f(Z) {cout << "D::f(Z) "; return this;}
    ~D() {cout << "~D ";}
    D() {cout << "D() ";}
```

```
};

class F: public B, public E, public D {
public:
    void f(bool) {cout << "F::f(bool) ";}
    F* f(Z) {cout << "F::f(Z) "; return this;}
    F() {cout << "F() ";}
    ~F() {cout << "~F ";}
```

```
};
```

```
class A {
public:
    void f(int) {cout << "A::f(int) "; f(true);}
    virtual void f(bool) {cout << "A::f(bool) ";}
    virtual A* f(Z) {cout << "A::f(Z) "; f(2); return this;}
    A() {cout << "A() ";}
```

```
class C: virtual public A {
public:
    C* f(Z) {cout << "C::f(Z) "; return this;}
    C() {cout << "C() ";}
```

```
class E: public C {
public:
    C* f(Z) {cout << "E::f(Z) "; return this;}
    ~E() {cout << "~E ";}
    E() {cout << "E() ";}
```

```
B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B* pbl= new F;
A* pal=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

A / F

```
class Z {
public: Z(int x) {}
};
```

DELTA PR1;

```
class B: virtual public A {
public:
void f(const bool&){cout<< "B::f(const bool&) ";}
void f(const int&){cout<< "B::f(const int&) ";}
virtual B* f(Z) {cout<< "B::f(Z) "; return this;}
virtual ~B() {cout<< "B ";}
B() {cout<< "B() ";}
};
```

```
class D: virtual public A {
public:
virtual void f(bool) const {cout<< "D::f(bool) ";}
A* f(Z) {cout<< "D::f(Z) "; return this;}
~D() {cout<< "D ";}
D() {cout<< "D() ";}
};
```

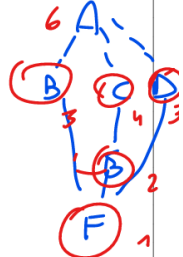
```
class E: public B, public D, public Z {
public:
void f(const bool&){cout<< "E::f(const bool&) ";}
F* f(Z){cout<< "E::f(Z) "; return this;}
F() {cout<< "E() ";}
~F() {cout<< "E ";}
};
```

```
class A {
public:
void f(int) {cout<< "A::f(int) "; f(true);}
virtual void f(bool) {cout<< "A::f(bool) ";}
virtual A* f(Z) {cout<< "A::f(Z) "; f(2); return this;}
A() {cout<< "A() ";}
};
```

```
class C: virtual public A {
public:
C* f(Z){cout<< "C::f(Z) "; return this;}
C() {cout<< "C() ";}
};
```

```
class E: public C {
public:
C* f(Z){cout<< "E::f(Z) "; return this;}
~E() {cout<< "E ";}
E() {cout<< "E() ";}
};
```

```
B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B* pbl= new F;
A* pal=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```



DISPER. VIRTUALE FN NS

LI STAMPANO TUTTI PER QUESTO

DISPER. (CONSUMI)

```
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;}
};
```

NON CONST  
NON COMPILATA

```
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;}
};
```

FN 5 -> 12 C)

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

B/K

J -> CONST

```
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;

### 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<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();}
};
```

OVERLOADS -> ANNOT.  
VIRTUAL -> POLIMORF.

POGLIO CONST  
CONST B\* p7 = NEW F() C  
C\* p7 = NEW F() D  
P7 -> K() F  
SO FINIAMO QUI  
STACK OVERFLOW  
-> CHIAM. INF.

```

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();

```

```

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();}
};

```

Handwritten notes and diagram:

- Red arrows point from `B::n` and `D::n` to `(dynamic_cast<D&>(p3->n()))g();`
- Diagram shows a hierarchy: B (circled) has children C and D (circled). C has child E (circled). E has child F (circled).

Si richiamano i seguenti fatti concernenti la libreria di I/O standard.

- `ios` è la classe base astratta e virtuale della gerarchia di tipi della libreria di I/O; la classe `istream` è derivata direttamente e virtualmente da `ios`; la classe `ifstream` è derivata direttamente da `istream`.
- `ios` rende disponibile un metodo costante e non virtuale `bool fail()` con il seguente comportamento: una invocazione `s.fail()` ritorna `true` se e solamente se lo stream `s` è in uno stato di fallimento (cioè, il failbit di `s` vale 1).
- `istream` rende disponibile un metodo non costante e non virtuale `long tellg()` con il seguente comportamento: una invocazione `s.tellg()`:
  - se `s` è in uno stato di fallimento allora ritorna -1;
  - altrimenti, cioè se `s` non è in uno stato di fallimento, ritorna la posizione della testina di input di `s`.
- `ifstream` rende disponibile un metodo costante e non virtuale `bool is_open()` con il seguente comportamento: una invocazione `s.is_open()` ritorna `true` se e solo se il file associato allo stream `s` è aperto.

Definire una funzione `long Fun(const ios& s)` con il seguente comportamento: una invocazione `Fun(s)`:

- se `s` è in uno stato di fallimento lancia un'eccezione di tipo `Failimento`, dove la classe `Failimento` va esplicitamente definita;
  - se `s` non è in uno stato di fallimento allora:
    - se `s` non è un `ifstream` ritorna -2;
    - se `s` è un `ifstream` ed il file associato non è aperto ritorna -1;
    - se `s` è un `ifstream` ed il file associato è aperto ritorna la posizione della cella corrente di input di `s`.
- Handwritten note: `throw Failimento()`, `if (s->fail())`

SOLUZIONE

```

② ifstream i = dynamic_cast<ifstream>(p3->n());
if (!i) return -2;
if (!i || !i->is_open()) return -1;
if (i && i->is_open()) i->tellg();

```

Gratias  
Randy!



Cuculo...)