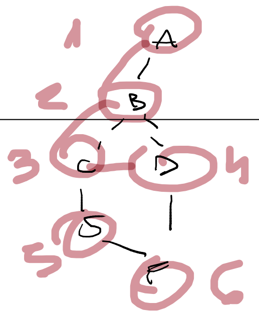
F* puntF = new F

DEFAULT

Esercizio Cosa Stampa



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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A "; }
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

```
class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

```
A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

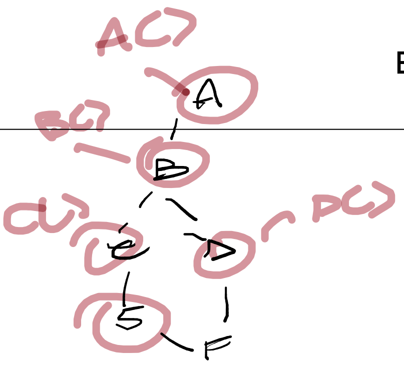
Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                          ...........................................................
02: E* puntE = new E(*pe);                     ...........................................................
03: pb3->f(3);                                 ...........................................................
04: pa->f(1.2);                                ...........................................................
05: pb1->f(Z(2));                              ...........................................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));   ...........................................................
07: static_cast<E*>(pc)->f();                  ...........................................................
08: pe->f(2);                                  ...........................................................
09: (pc->f(Z(3)))->f(4);                       ...........................................................
10: (pb3->f(Z(3)))->f(4);                      ...........................................................
11: delete pb3;                                ...........................................................
12: delete pe;                                 ...........................................................
```

E* puntE = new E(*pe);

Esercizio Cosa Stampa



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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

```
class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

COPIA

```
A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

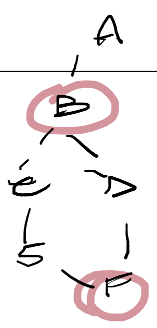
Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

01: `F* puntF = new F;`	....
02: `E* puntE = new E(*pe);`	....
03: `pb3->f(3);`	....
04: `pa->f(1.2);`	....
05: `pb1->f(Z(2));`	....
06: `if(typeid(pb3)==typeid(F)) pb3->f(Z(2));`	....
07: `static_cast<E*>(pc)->f();`	....
08: `pe->f(2);`	....
09: `(pc->f(Z(3)))->f(4);`	....
10: `(pb3->f(Z(3)))->f(4);`	....
11: `delete pb3;`	....
12: `delete pe;`	....

Esercizio Cosa Stampa

PB 3 ->f(3);



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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

1

2

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                          ....................................................
02: E* puntE = new E(*pe);                     ....................................................
03: pb3->f(3);                                 ....................................................
04: pa->f(1.2);                                ....................................................
05: pb1->f(Z(2));                              ....................................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));   ....................................................
07: static_cast<E*>(pc)->f();                  ....................................................
08: pe->f(2);                                  ....................................................
09: (pc->f(Z(3)))->f(4);                       ....................................................
10: (pb3->f(Z(3)))->f(4);                      ....................................................
11: delete pb3;                                ....................................................
12: delete pe;                                 ....................................................
```

PB1 -> F(Z(7));

Esercizio Cosa Stampa

$$\begin{tikzcd}
 & A \arrow[d, dash] & \\
 & B \arrow[dl, dash] \arrow[dr, dash] & \\
e \arrow[d, dash] & & D \arrow[d, dash] \\
5 \arrow[rr, dash] & & F
\end{tikzcd}$$

Z(7)

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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A "; }
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

```
class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

① ②

```
A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                          ..............................................
02: E* puntE = new E(*pe);                     ..............................................
03: pb3->f(3);                                 ..............................................
04: pa->f(1.2);                                ..............................................
05: pb1->f(Z(2));                              ..............................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));   ..............................................
07: static_cast<E*>(pc)->f();                  ..............................................
08: pe->f(2);                                  ..............................................
09: (pc->f(Z(3)))->f(4);                       ..............................................
10: (pb3->f(Z(3)))->f(4);                      ..............................................
11: delete pb3;                                ..............................................
12: delete pe;                                 ..............................................
```

TYPEID = UGUALI TIPI DINAMICI

F*

```
if(typeid(pb3)==typeid(F)) pb3->f(Z(2));
```

Esercizio Cosa Stampa

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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

$$\begin{tikzcd}
 & A \arrow[d, dash] & \\
 & B \arrow[dl, dash] \arrow[dr, dash] & \\
C \arrow[d, dash] & & D \arrow[d, dash] \\
E \arrow[rr, dash] & & F
\end{tikzcd}$$

TD

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

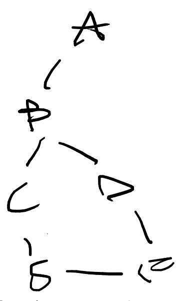
- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                             .........................................................
02: E* puntE = new E(*pe);                        .........................................................
03: pb3->f(3);                                    .........................................................
04: pa->f(1.2);                                   .........................................................
05: pb1->f(Z(2));                                 .........................................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));      .........................................................
07: static_cast<E*>(pc)->f();                     .........................................................
08: pe->f(2);                                     .........................................................
09: (pc->f(Z(3)))->f(4);                          .........................................................
10: (pb3->f(Z(3)))->f(4);                         .........................................................
11: delete pb3;                                   .........................................................
12: delete pe;                                    .........................................................
```

Esercizio Cosa Stampa

`static_cast<E*>(pc)->f();`

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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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



```
class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  ~~C* f(Z) {cout <<~~ "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

≠

```
A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                        ..............................................
02: E* puntE = new E(*pe);                   ..............................................
03: pb3->f(3);                               ..............................................
04: pa->f(1.2);                              ..............................................
05: pb1->f(Z(2));                            ..............................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2)); ..............................................
07: static_cast<E*>(pc)->f();                ..............................................
08: pe->f(2);                                ..............................................
09: (pc->f(Z(3)))->f(4);                     ..............................................
10: (pb3->f(Z(3)))->f(4);                    ..............................................
11: delete pb3;                              ..............................................
12: delete pe;                               ..............................................
```

PB -> F(Z); Z(INT)

Esercizio Cosa Stampa

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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

2 3

```
class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

1 2 3

F(B) -> [illegible]

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

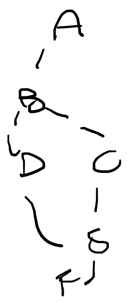
- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                        ..........................................
02: E* puntE = new E(*pe);                   ..........................................
03: pb3->f(3);                               ..........................................
04: pa->f(1.2);                              ..........................................
05: pb1->f(Z(2));                            ..........................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2)); ..........................................
07: static_cast<E*>(pc)->f();                ..........................................
08: pe->f(2);                                ..........................................
09: (pc->f(Z(3)))->f(4);                     ..........................................
10: (pb3->f(Z(3)))->f(4);                    ..........................................
11: delete pb3;                              ..........................................
12: delete pe;                               ..........................................
```

CONST INT&

Esercizio Cosa Stampa (pc->f(Z(3)))->f(4);

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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```



Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

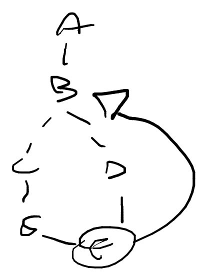
- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                          ..........................................................
02: E* puntE = new E(*pe);                     ..........................................................
03: pb3->f(3);                                 ..........................................................
04: pa->f(1.2);                                ..........................................................
05: pb1->f(Z(2));                              ..........................................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));   ..........................................................
07: static_cast<E*>(pc)->f();                  ..........................................................
08: pe->f(2);                                  ..........................................................
09: (pc->f(Z(3)))->f(4);                       ..........................................................
10: (pb3->f(Z(3)))->f(4);                      ..........................................................
11: delete pb3;                                ..........................................................
12: delete pe;                                 ..........................................................
```

```
(pb3->f(Z(3)))->f(4);
```

Esercizio Cosa Stampa

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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {                                        2
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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



```
class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };
                                                1
class F: public E, public D {
public:
  void f() const {cout << "F::f() ";}
  F* f(Z) {cout << "F::f(Z) "; return this;}
  void f(double) {cout << "F::f(double) ";}
  F() {cout << "F() "; }
  ~F() {cout << "~F ";}
};                                              3

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                          ..........................................................
02: E* puntE = new E(*pe);                     ..........................................................
03: pb3->f(3);                                 ..........................................................
04: pa->f(1.2);                                ..........................................................
05: pb1->f(Z(2));                              ..........................................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));   ..........................................................
07: static_cast<E*>(pc)->f();                  ..........................................................
08: pe->f(2);                                  ..........................................................
09: (pc->f(Z(3)))->f(4);                       ..........................................................
10: (pb3->f(Z(3)))->f(4);                      ..........................................................
11: delete pb3;                                ..........................................................
12: delete pe;                                 ..........................................................
```

Esercizio Cosa Stampa DELETE PB3;

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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

$$\begin{tikzcd} & A \arrow[d, dash] & \\ & B \arrow[dl, dash] \arrow[dr, dash] & \\ C \arrow[d, dash] & & D \arrow[dl, dash] \\ E \arrow[r, dash] & F & \end{tikzcd}$$

~B ~A

```
class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
 };

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

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                          ..............................................
02: E* puntE = new E(*pe);                     ..............................................
03: pb3->f(3);                                 ..............................................
04: pa->f(1.2);                                ..............................................
05: pb1->f(Z(2));                              ..............................................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));   ..............................................
07: static_cast<E*>(pc)->f();                  ..............................................
08: pe->f(2);                                  ..............................................
09: (pc->f(Z(3)))->f(4);                       ..............................................
10: (pb3->f(Z(3)))->f(4);                      ..............................................
11: delete pb3;                                ..............................................
12: delete pe;                                 ..............................................
```

Esercizio Cosa Stampa

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

class A {
public:
  A() {cout << "A() "; }
  ~A() {cout << "~A ";}
};

class B: public A {
 public:
  void f(int) {cout << "B::f(int) "; f(3.14); }
  virtual void f(double) {cout << "B::f(double) ";}
  virtual B* f(Z) {cout << "B::f(Z) "; return this; }
  B() {cout << "B() "; }
  ~B() {cout << "~B ";}
};

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

```
class D: virtual public B {
public:
  D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
  virtual void f(double) {cout << "D::f(double) ";}
  D() {cout << "D() ";}
  ~D() {cout << "~D ";}
};

class E: public C {
public:
  virtual void f() {cout << "E::f() "; C::f(Z(1));}
  C* f(Z) {cout << "E::f(Z) "; f(); return this;}
  E() {cout << "E() "; }
  E(const E& e) {cout << "Ec ";}
  ~E() {cout << "~E ";}
};

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

```
A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```

$$\begin{tikzcd} & A \arrow[d, dash] & \\ & B \arrow[dl, dash] \arrow[dr, dash] & \\ C \arrow[d, dash] & & D \arrow[dl, dash] \\ E \arrow[r, dash] & F & \end{tikzcd}$$

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED BEHAVIOUR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva **chiaramente** la stampa che l'esecuzione produce in output su `cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
01: F* puntF = new F;                              ..............................
02: E* puntE = new E(*pe);                         ..............................
03: pb3->f(3);                                     ..............................
04: pa->f(1.2);                                    ..............................
05: pb1->f(Z(2));                                  ..............................
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));       ..............................
07: static_cast<E*>(pc)->f();                      ..............................
08: pe->f(2);                                      ..............................
09: (pc->f(Z(3)))->f(4);                           ..............................
10: (pb3->f(Z(3)))->f(4);                          ..............................
11: delete pb3;                                    ..............................
12: delete pe;                                     ..............................
```