Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

Nome..... Cognome..... Matricola.....

Esercizio Cosa Stampa

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

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


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

```
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 ";}
};
```

```
B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

Nome..... Cognome..... Matricola.....

Esercizio Cosa Stampa

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

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

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

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 ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

Nome..... Cognome..... Matricola.....

Esercizio Cosa Stampa

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

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

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

```
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 ";}
};
```

```
B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

Nome..... Cognome..... Matricola.....

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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

Nome..... Cognome..... Matricola.....

Esercizio Cosa Stampa

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

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

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

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 ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

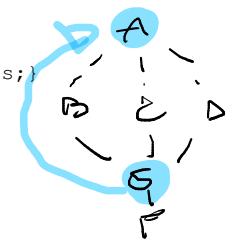
Nome..... Cognome..... Matricola.....

Esercizio Cosa Stampa

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

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

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

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 ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

(pa4→f(Z(3)))→f(4);

A /B



Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

Nome..... Cognome..... Matricola.....

Esercizio Cosa Stampa

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

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

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

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 ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

Nome..... Cognome..... Matricola.....

Esercizio Cosa Stampa

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

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

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

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 ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E;
F* pf = new F; B *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 12/7/2022

Nome..... Cognome..... Matricola.....

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 *pb1= new F;
A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe, *pa5=pf;
```

Se distruttore virtuale in classe base ⟶ va giù!

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behaviour o errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

```
pa3->f(3);                          .....

pa5->f(3);                          .....

pb1->f(true);                       .....

pa4->f(true);                       .....

pa2->f(Z(2));                       .....

pa5->f(Z(2));                       .....

(dynamic_cast<E*>(pa4))->f(Z(2));   .....

(dynamic_cast<C*>(pa5))->f(Z(2));   .....

pb->f(3);                           .....

pc->f(3);                           .....

(pa4->f(Z(3)))->f(4);               .....

(pc->f(Z(3)))->f(4);                .....

E* puntE = new F;                   .....

delete pa5;                         .....

delete pb1;                         .....
```

Esercizio Funzione

Si ricordano le seguenti specifiche riguardanti la libreria standard di I/O.

(1) La classe `ios` è la classe base polimorfa della gerarchia di tipi per l'I/O. Un oggetto di `ios` rappresenta un generico stream. Lo stato di uno stream è un intero in [0,7], dove lo stato 0 significa stato privo di errori, mentre lo stato 2 significa stato di fallimento recuperabile. `ios` rende disponibile un metodo costante `int rdstate()` con il comportamento: `s.rdstate()` ritorna lo stato di `s`. Inoltre `ios` rende disponibile un metodo `void setstate(int)` con il comportamento: `s.setstate(x)` modifica al valore `x` lo stato di `s`.

(2) La classe `istream` è derivata direttamente e virtualmente da `ios` ed i suoi oggetti rappresentano un generico stream di input. La classe `istream` rende disponibile un metodo costante `int tellg()` con il seguente comportamento: `i.tellg()` ritorna la posizione della testina di input nello stream di input `i`.

(3) La classe `ostream` è derivata direttamente e virtualmente da `ios` ed i suoi oggetti rappresentano un generico stream di output. La classe `ostream` rende disponibile un metodo costante `int tellp()` con il seguente comportamento: `o.tellp()` ritorna la posizione della testina di output nello stream di output `o`.

(4) La classe `iostream` è derivata direttamente per ereditarietà multipla da `istream` e `ostream` ed i suoi oggetti rappresentano uno stream di input/output.

(5) La classe `fstream` è derivata direttamente da `iostream` ed i suoi oggetti rappresentano un generico file stream di I/O. La classe `fstream` rende disponibile un metodo costante `bool is_open()` con il seguente comportamento: `f.is_open()` ritorna true se il file stream `f` è associato a qualche file, altrimenti ritorna false. Inoltre la classe `fstream` rende disponibile un metodo `void close()` con il seguente comportamento: `f.close()` disassocia il file correntemente associato al file stream `f`, mentre se `f` non è associato ad alcun file allora non provoca effetti.

Definire una funzione `vector<fstream*> Fun(const vector<const ios*>&)` con il seguente comportamento: in ogni invocazione `Fun(v)` la funzione deve soddisfare le seguenti specifiche:

(a) a tutti gli stream di input/output puntati da un puntatore contenuto nel vector `v` che abbiano la posizione della testina di input maggiore della posizione della testina di output, modifica lo stato in uno stato di fallimento recuperabile.

(b) ritorna un vector di puntatori a file stream contenente tutti e soli i puntatori a file stream contenuti nel vector `v` che sono in uno stato privo di errori e che hanno un qualche file a loro associato; tali file stream devono inoltre essere disassociati al file a loro associato. Se invece non vi è nessun file stream privo di errori e che ha un qualche file associato allora viene sollevata una eccezione di tipo `std::exception`.

SOLUZIONE