Laurea in Informatica - Programmazione ad Oggetti - Appello d'Esame 22/6/2022

XX		3.6 . 1 . 1
Nome	Cognome	Matricola
1 101110	Cognome	141411111111111111111111111111111111111

Esercizio Funzione

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.tellq():
 - 1. se s è in uno stato di fallimento allora ritorna -1;
 - 2. 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&) con il seguente comportamento: una invocazione Fun (s):

- (1) se s è in uno stato di fallimento lancia una eccezione di tipo Fallimento, dove la classe Fallimento va esplicitamente definita;
- (2) se s non è in uno stato di fallimento allora:
 - (a) se s non è un ifstream ritorna -2;
 - (b) se s è un ifstream ed il file associato non è aperto ritorna -1;
 - (c) se s è un ifstream ed il file associato è aperto ritorna la posizione della cella corrente di input di s.

SOLUZIONE	
class Fallimento{	
private:	•
·····std::string·msg;··································	
public:	
Fallimento(std::string m): msg(m) {} std::string getMsg() const{	•
return msg; }	
3;	•
long fun(const ios& s){	•
if(s.fail()) // mettiamo il punto perché è un riferimento	•
// se tu avessi const ios* s,	
······//allora faresti (*s).fail() oppure s→fail()······	
throw Fallimento("");	
: (-t	•
<pre>ifstream* if = dynamic_cast<ifstream*>(&s); if(!if) return -2;</ifstream*></pre>	
if(if && !if→is_open()) return -1;	
if(if && :if→is_open()) s:tellg();	
•	
	•

```
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;}</pre>
                                                                                  P4-7F();
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
                                                                         class D: virtual public B {
class C: virtual public B {
public:
                                                                         public:
  C() {cout<< " C() ";}
                                                                           D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                           ~D() {cout<< " ~D() ";}
                                                                          virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                          const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                          void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                          void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                        class F: public E {
public:
                                                                        public:
  E() {cout<< " E() ";}</pre>
                                                                           F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                           ~F() {cout<< " ~F() ";}
 virtual void g() const {cout <<" E::g
const E* j() {cout << (E::j)"; return</pre>
                                                                          F(const F& x): B(x) {cout<< " Fc ";}
                                                                          void k() {cout <<" F::k "; g();}</pre>
                                                                          void m() {cout <<" F::m "; j();}</pre>
  void m() {cout <<" E::m</pre>
                                 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 = \text{new D}(); const B*p6 = \text{new E}(); const B*p7 = \text{new F}(); F f;
```

- 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 behavioùr o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f();

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

p3->k();

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

(dynamic_cast<D&>(p3->n()).g();

p2->f();

p2->m();

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

(p5->n()).g();

F x;

C* p = new F(f);

p1->m();

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

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

(dynamic_cast<Cent F*>(p1->j()))->k();

(dynamic_cast<Cent F*>(p1->j()))->k();

(dynamic_cast<Cent F*>(p1->j()))->k();

(dynamic_cast<Cent F*>(p1->j()))->k();
```

```
class B {
public:
  B() {cout << "B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                               (P4.NO)>>M).
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout \( " B::n "; return *this; \)}
                                                                        class D: virtual public B {
class C: virtual public B
public:
                                                                        public:
  C() {cout<< " C() ";}
                                                                          D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                          ~D() {cout<< " ~D() ";}
                                                                         virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout << " C::g ";}</pre>
  void k() override {cout <<" C::k "; B::n();}</pre>
                                                                          const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                          void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                          void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D
                                                                        class F: public E {
public:
                                                                        public:
  E() {cout<< " E() ";}
                                                                          F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                          ~F() {cout<< " ~F() ";}
  virtual void g() const {cout
                                                                          F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E:
                                    return this; }
                                                                         void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; q
                                                                         void m() {cout <<" F::m "; j();}</pre>
                                  (); 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()
```

- 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 behavioùr o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D&>(p3->n())).g();
p2->f();
p2->m();
(p5->n()).g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<Cost F*>(p1->j())->k();
(dynamic_cast<C*>(const_cast<B*>(p7)))->k();
delete p7;
```

```
class B {
public:
  B() {cout << "B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "</pre>
  virtual void g() const {cout <<" \( \):
                                                                                          P3>K()
  virtual const B* j() {cout<< "B:j"; return the
virtual void k() {cout < "B:k"; j(); m(); }</pre>
  virtual void k() {coat
void m() {cout <<" B: m</pre>
                               "; g(); j();}
  virtual B& n() {cout K<" B::n "; return *this;}</pre>
                                                                              class D: virtual public B {
class C: virtual public
public:
                                                                              public:
  C() {cout<< " C() ";}
                                                                                D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                                ~D() {cout<< " ~D() ";}
                                                                                virtual void g() {cout <</p>
const B* j() {cout <</pr>
D::j
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                                const B* j() {cout <</
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                                void k() const {cout <<"
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                                void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                              class F: public E {
public:
                                                                              public:
  E() {cout<< " E() ";}
                                                                                F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                                ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                                F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                                void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                                void m() {cout <<" F::m "; j();}</pre>
  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 <math>B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

- 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 behavioùr o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f();

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

p3->k();

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

(dynamic_cast<D6>(p3->n()).g();

p2->f();

p2->m();

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

(p5->n()).g();

F x;

C* p = new F(f);

p1->m();

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

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

(dynamic_cast<E*>(p6))->j();

(dynamic_cast<C*>(const_cast<B*>(p7)))->k();

delete p7;
```

```
class B {
public:
  B() {cout << "B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                (P3.N()).M().
  virtual void f() {cout << " B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g=";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<"_B::m_"; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
                                                                       D() {cout<< " D() ";}
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
                                                                       ~D() {cout<< " ~D() ";}
                                                                      virtual void g() {cout <<" \mathbb{f}::g
  virtual void g() const override {cout <<" C::g ";}</pre>
                                                                                                    i "; return this;}
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                       const B* j() {cout <<" D::
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                       void k() const {cout << " D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}
                                                                       void m() {cout <<" D::m "; g(); j();}
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                    public:
  E() {cout<< " E() ";}
                                                                      F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                       ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                      F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                      void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                      void m() {cout <<" F::m "; j();}</pre>
  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;
```

- 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 behavioùr o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D&>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Cest F*>(p1->j())->g();
(dynamic_cast<Cest F*>(p1->j())->k();
(dynamic_cast<C*>(const_cast<B*>(p7)))->k();
delete p7;
```

```
class B {
public:
  B() {cout << " B() ";}
                                                                     Dx P3 = NOW (P3->NC) . G()
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
lass C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
  C() {cout<< " C() ";}
                                                                       D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                        ~D() {cout<< " ~D() ";}
                                                                       virtual void g() {cout <<" D:':g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                       const B* j() {cout <<" D::j"; return this;}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                       void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                       void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                     public:
  E() {cout<< " E() ";}
                                                                       F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                        ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                       F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                       void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                       void m() {cout <<" F::m "; j();}</pre>
  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 = \text{new D}(); const B * p6 = \text{new E}(); const B * p7 = \text{new F}(); F f;
```

- 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 behavioùr o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
p4->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<Cost F*>(p1->j())->g();
(dynamic_cast<C*>(const_cast<B*>(p7)))->k();
(dynamic_cast<C*>(const_cast<B*>(p7)))->k();
delete p7;
```

```
class B {
public:
  B() {cout << " B() ";}
  virtual ~B() {cout<< " ~B() ";},</pre>
                                                                                       P2 >FL).
  virtual void f() {cout <<" B::f "; g(); j()</pre>
  virtual void g() const {cout << " B::g
  virtual const B* j() {cout<<" B*:j"; return this;}
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                        class D: virtual public B {
public:
                                                                        public:
  C() {cout<< " C() ";}
                                                                          D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                          ~D() {cout<< " ~D() ";}
                                                                         virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout << " C;;g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                          const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                         void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                         void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                        class F: public E {
public:
                                                                       public:
  E() {cout<< " E() ";}
                                                                          F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                          ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}</pre>
                                                                         F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                         void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                         void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}</pre>
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B * p5 = \text{new } \overline{D(1; \text{const } B * p6} = \text{new } E(); \text{const } B * p7 = \text{new } F(); \text{ } f;
```

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

```
p4->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();

p2->m();
(p2->j())->g();
(p5->n()).g();

F x;

C* p = new F(f);
p1->m();
(p1->j())->k();
(dynamic_cast<Cnst F*>(p1->j()))->g();
(dynamic_cast<Cr>
(dynamic_cast<Cr>
(dynamic_cast<Cr>
(const_cast<B*>(p7)))->k();
(dynamic_cast<Cr>
(dynamic_cast<Cr>
(const_cast<B*>(p7)))->k();
(delete p7;
```

```
class B {
public:
                                                                                           Fx.
 B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout <<" B::n "; return *this;}</pre>
class C: virtual public B {
                                                                       class D: virtual public B {
public:
                                                                       public:
  C() {cout<< " C() ";}
                                                                         D() {cout<< " D() ";}
   C() {cout<< " ~C() ";}
                                                                         ~D() {cout<< " ~D() ";}
                                                                         virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                         const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                         void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                         void m() {cout <<" D::m "; g(); j();}</pre>
};
class E: public C, public D {
                                                                       class F: public E {
public:
                                                                       public:
                                                                         F() {cout<< " F() ";}
  E() {cout << " E() ";}
  ~E() {cout<< " ~E() ";}
                                                                         ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                         F(const F& x): B(x) {cout<< " Fc ";}</pre>
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                        void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                        void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}</pre>
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B * p5 = \text{new D()}; const B * p6 = \text{new E()}; const B * p7 = \text{new F()}; F f;
```

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

```
p4->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D&>(p3->n()).g();
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;
C* p = new F(f);
p1->m();
(p1->j())->k();
(dynamic_cast<Cost F*>(p1->j()))->g();
(dynamic_cast<Cost F*>(p6))->j();
(dynamic_cast<C>(const_cast<B>(p7)))->k();
delete p7;
```

```
class B {
                                                                    C* P= NOW F(F);
CORACHIA + A
public:
  B() {cout << " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
                                                            F ( CONST FOR X)
class C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
  C() {cout<< " C() ";}
                                                                       D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                       ~D() {cout<< " ~D() ";}
                                                                       virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                       const B* j() {cout <<" D::j "; return this;}</pre>
                                                                       void k() const {cout <<" D::k "; k();}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
  B& n() override {cout << " C::n "; return *this;}
                                                                       void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                     public:
  E() {cout<< " E() ";}
                                                                       F() {cout<< "F() ";}
                                                                     F() {cout << " F() ";}
F(const F& x): B(x) cout << " Fc ";}
void k() {cout
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                                            F::m "; j();}
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                       void m() {cout <<'</pre>
  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;
```

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

```
p4->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();
p2->m();
(p2->j())->g();
F x;

C* p = new F(f);
p1->m();
(dynamic_cast<Ce>(p6)->j())->g();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(delete p7;
```

```
class B {
public:
  B() {cout << "B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
                                                                                 (P1 > 5()) > KC);
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
                                                                        class D: virtual public B {
class C: virtual public B {
public:
                                                                        public:
  C() {cout<< " C() ";}
                                                                          D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                          ~D() {cout<< " ~D() ";}
                                                                         virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout << " C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                          const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                          void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                          void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                        class F: public E {
public:
                                                                        public:
  E() {cout<< " E() ";}</pre>
                                                                          F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                          ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}</pre>
                                                                          F(const F& x): B(x) {cout<< " Fc ";}</pre>
(Tons) E* j() {cout <<" E::j "; return this;}
                                                                          void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                          void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}
      = \text{new } E(); B* p2 = \text{new } C(); B* p3 = \text{new } D(); C* p4 = \text{new } E();
const B \star p5 = \text{new D}(); const B \star p6 = \text{new E}(); const B \star p7 = \text{new F}(); F f;
```

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

```
p4->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast<D6>(p3->n())).g();
p2->f();

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

f x;

C+ p = new F(f);
p1->m();
(dynamic_cast<Ce>(p6)->j())->g();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(dynamic_cast<Ce>(const_cast<Be>(p7)))->k();
(delete p7;
```

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
 virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
  C() {cout<< " C() ";}
                                                                       D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                       ~D() {cout<< " ~D() ";}
                                                                      virtual void g() {cout <<" D::g ";}</pre>
 virtual void g() const override {cout << " C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                       const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                       void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                       void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                     public:
  E() {cout<< " E() ";}
                                                                       F() {cout << " F() ";}
  ~E() {cout<< " ~E() ";}
                                                                       ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}</pre>
                                                                       F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                       void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                       void m() {cout <<" F::m "; j();}</pre>
 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;
```

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

p4->f();
(p4->n()).m();
p3->k();
(p3->n()).m();
(dynamic_cast <d&>(p3->n())).g();</d&>
p2->f();
p2->m();
(p2->j())->g();
(p5->n()).g();
F x;
C* p = new F(f);
p1->m();
(p1->j())->k(); PANGSA POL FAN
(dynamic_cast <const f*="">(p1->j()))->g();</const>
(dynamic_cast <e*>(p6))->j();</e*>
(dynamic_cast <c*>(const_cast<b*>(p7)))->k();</b*></c*>
delete p7;

```
class B {
                                                                           1 UNDSPIRED
public:
  B() {cout << " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
                                                                  DYN_CAST (LONS) Fx>(p1 >> 0)
S::3 .... (x)
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
  C() {cout<< " C() ";}
                                                                      D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                       ~D() {cout<< " ~D() ";}
                                                                      virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
                                                                       const B* j() {cout <<" D::j "; return this;}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
  virtual void m() {cout << " C::m "; g(); j();}</pre>
                                                                      void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                      void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                    public:
  E() {cout<< " E() ";}
                                                                      F() {cout<< "F() ";}
                                                                       ~F() {cout<< " ~F() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}</pre>
                                                                      F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                      void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                      void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}</pre>
     = 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;
```

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

```
p4->f();

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

p3->k();

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

(dynamic_cast<D6>(p3->n()).g();

p2->f();

p2->m();

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

(p5->n()).g();

F x;

C+ p = new F(f);

p1->m();

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

(dynamic_cast<Cost Const_cast<B*>(p7)))->k();

delete p7;
```

```
class B {
public:
  B() {cout << " B() ";}
  virtual ~B() {cout<< " ~B() ";}</pre>
  virtual void f() {cout <<" B::f "; g(); j();}</pre>
  virtual void g() const {cout <<" B::g ";}</pre>
  virtual const B* j() {cout<<" B::j "; return this;}</pre>
  virtual void k() {cout <<" B::k "; j(); m(); }</pre>
  void m() {cout <<" B::m "; g(); j();}</pre>
  virtual B& n() {cout << " B::n "; return *this;}</pre>
class C: virtual public B {
                                                                     class D: virtual public B {
public:
                                                                     public:
  C() {cout<< " C() ";}
                                                                       D() {cout<< " D() ";}
  ~C() {cout<< " ~C() ";}
                                                                       ~D() {cout<< " ~D() ";}
                                                                      virtual void g() {cout <<" D::g ";}</pre>
  virtual void g() const override {cout <<" C::g ";}</pre>
  void k() override {cout << " C::k "; B::n();}</pre>
                                                                       const B* j() {cout <<" D::j "; return this;}</pre>
  virtual void m() {cout <<" C::m "; g(); j();}</pre>
                                                                       void k() const {cout <<" D::k "; k();}</pre>
  B& n() override {cout << " C::n "; return *this;}</pre>
                                                                       void m() {cout <<" D::m "; g(); j();}</pre>
class E: public C, public D {
                                                                     class F: public E {
public:
                                                                     public:
  E() {cout<< " E() ";}
                                                                       F() {cout<< "F() ";}
  ~E() {cout<< " ~E() ";}
                                                                       ~F() {cout<< " ~F() ";}
  virtual void g() const {cout <<" E::g ";}
                                                                      F(const F& x): B(x) {cout<< " Fc ";}
  const E* j() {cout <<" E::j "; return this;}</pre>
                                                                       void k() {cout <<" F::k "; g();}</pre>
  void m() {cout <<" E::m "; g(); j();}</pre>
                                                                       void m() {cout <<" F::m "; j();}</pre>
  D& n() final {cout <<" E::n "; return *this;}</pre>
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;
```

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

```
p4->f();

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

p3->k();

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

(dynamic_cast<D6>(p3->n()).g();

p2->f();

p2->m();

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

(p5->n()).g();

F x;

C* p = new F(f);

p1->m();

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

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

(dynamic_cast<E*>(p6))->j();

(dynamic_cast<C*>(const_cast<B*>(p7)))->k();

delete p7;
```