Esercizio 2

```
class B {
public:
  int x;
  B(int z=1): x(z) {}
  virtual void f() const {cout << x << " B::f() ";}
  virtual bool operator<=(const B& r) const {return true;}
};

class C: virtual public B {
public:
  virtual void f() const {cout << "C::f() ";}
  virtual bool operator<=(const B& r) const { return dynamic_cast<const C*>(&r) != 0 ? true : false; }
};

class D: virtual public B {
public:
  virtual void g() const {cout << "D::g() ";}
  virtual void h() const {cout << "D::h() "; g(); }
  virtual bool operator<=(const B& r) const { return dynamic_cast<const D*>(&r) != 0 ? true : false; }
};

class E: public D {
public:
  virtual void f() const {cout << "E::f() ";}
  virtual void h() const {cout << "E::h() ";}
  virtual bool operator<=(const B& r) const { return dynamic_cast<const E*>(&r) != 0 ? true : false; }
};

class F: public C, public E {
public:
  F(): B(2) {}
  virtual void f() const {cout << x << " F::f() ";}
  virtual void g() const {cout << "F::g() "; D::g(); }
  virtual bool operator<=(const B& r) const { return dynamic_cast<const F*>(&r) != 0 ? true : false; }
};

void fun(const vector<B*>& v) {
  D* p;
  for(int k=0; k != v.size(); ++k) {
    v[k]->f();
    if( k+1 < v.size() && *v[k] <= *v[k+1] ) v[k+1]->f();
    p = dynamic_cast<E*>(v[k]);
    if(p) { static_cast<E*>(p)->g(); p->h(); }
    cout << "**" << k << endl;
  }
}
int main() {
  B b; C c; D d; E e; F f;
  vector<B*> v = {&d, &f, static_cast<D*>(&e), &d, &c, &d, &d, &b, &b, &c, &b, &f, static_cast<E*>(&f), &e, &c};
  fun(v);
}
```

Si considerino le precedenti definizioni, la cui compilazione (in C++11 con gli opportuni `include` e `using`) **non** provoca errori. Si sc
spazio delimitato dai puntini le stampe prodotte in output su `cout` dall'esecuzione del `main()`: per ogni iterazione `k` del ciclo `for` scrive
stampa nella riga `**k` (o **NESSUNA STAMPA** se non ve ne sono), mentre se per una iterazione `k` l'esecuzione provoca un undefined beh
si scriva **UNDEFINED**, e si considerino le successive iterazioni come se non l'undefined behaviour non si fosse verificato.

.....

① -> B::F (IN D NO F()) (D, F)

SE F ≤ D -> F::F

2° PARTE SE V[K] ≤ E -> NON LO È!

② (F, STATIC_CAST<D*>(E));

```
class E: public D {
public:
  virtual void f() const {cout << "E::f() ";}
  virtual void h() const {cout << "E::h() ";}
  virtual bool operator<=(const B& r) const { return dynamic_cast<const E*>(&r) != 0 ? true : false; }
};

class F: public C, public E {
public:
  F(): B(2) {}
  virtual void f() const {cout << x << " F::f() ";}
  virtual void g() const {cout << "F::g() "; D::g(); }
  virtual bool operator<=(const B& r) const { return dynamic_cast<const F*>(&r) != 0 ? true : false; }
};

void fun(const vector<B*>& v) {
  D* p;
  for(int k=0; k != v.size(); ++k) {
    v[k]->f();
    if( k+1 < v.size() && *v[k] <= *v[k+1] ) v[k+1]->f();
    p = dynamic_cast<E*>(v[k]);
    if(p) { static_cast<E*>(p)->g(); p->h(); }
    cout << "**" << k << endl;
  }
}
int main() {
  B b; C c; D d; E e; F f;
  vector<B*> v = {&d, &f, static_cast<D*>(&e), &d, &c, &d, &d, &b, &b, &c, &b, &f, static_cast<E*>(&f), &e, &c};
  fun(v);
}
```

① F::F / D::G / E::H

(3)

```
void fun(const vector<B*>& v) {
  D* p;
  for(int k=0; k != v.size(); ++k) {
    v[k]->f();
    if( k+1 < v.size() && *v[k] <= *v[k+1] ) v[k+1]->f();
    p = dynamic_cast<E*>(v[k]);
    if(p) { static_cast<E*>(p)->g(); p->h(); }
    cout << "**" << k << endl;
  }
}
int main() {
  B b; C c; D d; E e; F f;
  vector<B*> v = {&d, &f, static_cast<D*>(&e), &d, &c, &d, &d, &b, &b, &c, &b, &f, static_cast<E*>(&f), &e, &c};
  fun(v);
}
```

Si considerino le precedenti definizioni, la cui compilazione (in C++11 con gli opportuni `include` e `using`) **non** provoca errori. Si sc

(4)

```
};

class F: public C, public E {
public:
  F(): B(2) {}
  virtual void f() const {cout << x << " F::f() ";}
  virtual void g() const {cout << "F::g() ";  D::g(); }
  virtual bool operator<=(const B& r) const { return dynamic_cast<const F*>(&r) != 0 ? true : false; }
};

void fun(const vector<B*>& v) {
  D* p;
  for(int k=0; k != v.size(); ++k) {
    v[k]->f();
    if( k+1 < v.size() && *v[k] <= *v[k+1] ) v[k+1]->f();
    p = dynamic_cast<E*>(v[k]);
    if(p) { static_cast<E*>(p)->g(); p->h(); }
    cout << "**" << k << endl;
  }
}
int main() {
  B b; C c; D d; E e; F f;
  vector<B*> v = {&d, &f, static_cast<D*>(&e), &d, &c, &d, &d, &b, &b, &c, &b, &f, static_cast<E*>(&f), &e, &c};
  fun(v);
}
```

(5)

```
void fun(const vector<B*>& v) {
  D* p;
  for(int k=0; k != v.size(); ++k) {
    v[k]->f();
    if( k+1 < v.size() && *v[k] <= *v[k+1] ) v[k+1]->f();
    p = dynamic_cast<E*>(v[k]);
    if(p) { static_cast<E*>(p)->g(); p->h(); }
    cout << "**" << k << endl;
  }
}
int main() {
  B b; C c; D d; E e; F f;
  vector<B*> v = {&d, &f, static_cast<D*>(&e), &d, &c, &d, &d, &b, &b, &c, &b, &f, static_cast<E*>(&f), &e, &c};
  fun(v);
}
```

(6)

```
};

class E: public D {
public:
  virtual void f() const {cout << "E::f() ";}
  virtual void h() const {cout << "E::h() ";}
  virtual bool operator<=(const B& r) const { return dynamic_cast<const E*>(&r) != 0 ? true : false; }
};

class F: public C, public E {
public:
  F(): B(2) {}
  virtual void f() const {cout << x << " F::f() ";}
  virtual void g() const {cout << "F::g() ";  D::g(); }
  virtual bool operator<=(const B& r) const { return dynamic_cast<const F*>(&r) != 0 ? true : false; }
};

void fun(const vector<B*>& v) {
  D* p;
  for(int k=0; k != v.size(); ++k) {
    v[k]->f();
    if( k+1 < v.size() && *v[k] <= *v[k+1] ) v[k+1]->f();
    p = dynamic_cast<E*>(v[k]);
    if(p) { static_cast<E*>(p)->g(); p->h(); }
    cout << "**" << k << endl;
  }
}
int main() {
  B b; C c; D d; E e; F f;
  vector<B*> v = {&d, &f, static_cast<D*>(&e), &d, &c, &d, &d, &b, &b, &c, &b, &f, static_cast<E*>(&f), &e, &c};
  fun(v);
}
```