Si considerino le seguenti definizioni.

```
class B {
private:
  vector<bool>* ptr;
  virtual void m() const =0;
};

class D: public B {
private:
  int x;
};

class F: public D {
private:
  list<int*> l;
  int& ref;
  double* p;
public:
  void m() const {}
  // ridefinizione del costruttore di copia di F
};
```

```
// Copia profonda (standard)

F(const F& f): D(f), l(f.l), ref(f.ref), p(f.p) {}

// Assegnazione profonda (standard)

F& operator=(const F& f){
        D::operator=(f);

        l = f.l;
        ref = f.ref;
        p = f.p;

        return *this;
}
```

Ridefinire il costruttore di copia della classe `F` in modo tale che il suo comportamento coincida con quello del costruttore di copia standard di `F`.

Esercizio Definizioni

```
class Z {
private:
  int x;
};

class B {
private:
  Z bz;
};

class C: virtual public B {
private:
  Z cz;
};

class D: public C {
};

class E: virtual public B {
public:
  Z ez;
  // ridefinizione assegnazione
  // standard di E
};

class F: public D, public E {
private:
  Z* fz;
public:
  // ridefinizione del costruttore di copia profonda di F
  // ridefinizione del distruttore profondo di F
  // definizione del metodo di clonazione di F
};
```

```
E& operator=(const E& e){
        B::operator=(e);
        ez = e.ez;
        return *this;
}
```

```
F(const F& f): B(f), D(f), E(f), fz(f.fz) {}
~F() {if(fz) delete fz;}
virtual F* clone() const {return new F(*this); }
```

Si considerino le definizioni sopra.
(1) Ridefinire l'assegnazione della classe `E` in modo tale che il suo comportamento coincida con quello dell'assegnazione standard di `E`. Naturalmente non è permesso l'uso della keyword `default`.
(2) Ridefinire il costruttore di copia profonda della classe `F`.
(3) Ridefinire il distruttore profondo della classe `F`.
(4) Definire il metodo di clonazione della classe `F`.