

Esercizio Gerarchia

Definire una **unica gerarchia di classi** che includa:

- (1) una unica classe base polimorfa A alla radice della gerarchia;
- (2) una classe derivata astratta B;
- (3) una sottoclasse C di B che sia concreta;
- (4) una classe D che non permetta la costruzione pubblica dei suoi oggetti, ma solamente la costruzione di oggetti di D che siano sottooggetti;
- (5) una classe E definita mediante **derivazione multipla a diamante con base virtuale**, che abbia D come supertipo, e con l'**assegnazione ridefinita pubblicamente con comportamento identico a quello dell'assegnazione standard** di E.

// Gerarchia 1

```
class A{
    virtual ~A(){};
};

class B: public A{
    // astratto = virtuale puro
    // classe astratta = non poter instanziar oggetti
    virtual double calcola() = 0;
};

class E: public D{
    private:
        int i;
        double* d;
    public:
        E& operator=(const E& e){
            D::operator=(e);
            i = e.i;
            d = e.d;

            return *this;
        }
};
```

DIAMOND PROBLEM



```
class C: public B{
    // concreto = utilizzo il metodo astratto
    // e gli "do un senso" = lo uso
    virtual double calcola(){
        return 42;
    }
};

class D: virtual public B, virtual public C{
    private:
        int x;
    protected:
        D(int x1): x(x1) {};
};
```

Esercizio 2

Scrivere un programma che consista esattamente di tre classi A, B e C, dove B è un sottotipo di A, mentre C non è in relazione di subtyping nè con A nè con B, che dimostri in un metodo di C un tipico esempio di un **uso giustificato e necessario** della conversione di tipo `dynamic_cast` per effettuare type downcasting. A questo fine, si usino il **minor numero possibile di metodi**.



```
class A {};
class B: public A{};

class C{
    // Giusta
    A* a;
    C* c = dynamic_cast<C*>(a);

    // Caso da coprire
    B* b = dynamic_cast<B*>(a);
    C* c = dynamic_cast<C*>(b);
};
```

```

class A {
public:
    virtual ~A() {}
};

class B: public A {};
class C: public A {};
class D: public C {};

template<class T>
void Fun(T* pt){
    bool b=0;
    try{ throw T(*pt); }
    catch(B){cout << "B "; b=1;}
    catch(C){cout << "C "; b=1;}
    catch(D){cout << "D "; b=1;}
    catch(A){cout << "A "; b=1;}
    if(!b) cout << "NO ";
}

B b; C c; D d; A *pa1 = &b, *pa2 = &d;
B *pb1 = dynamic_cast<B*>(pa1); B *pb2 = dynamic_cast<B*>(pa2);

```

$\text{FUN}(\text{D})(\text{PA2}) \rightarrow \text{NC}$
 $\rightarrow \text{A/D?}$

$\text{FUN}(\text{C})(\text{PA2})$
 $\rightarrow \text{C}$

$\text{FUN}(\text{D})(\text{C}) \rightarrow \text{C}$
 $\text{FUN}(\text{C})(\text{C}) \rightarrow \text{C}$
 $\text{FUN}(\text{A})(\text{PB1}) \rightarrow \text{A}$
 $\text{FUN}(\text{PA1}) \rightarrow \text{A}$
 $\text{FUN}(\text{PA2}) \rightarrow \text{A}$] TS = TIPO
 $\text{FUN}(\text{PB1}) \rightarrow \text{B}$
 $\text{FUN}(\text{PB2}) \rightarrow \text{B}$
 $\text{FUN}(\text{C})(\text{PA2}) \rightarrow \text{A}$
 $\text{FUN}(\text{B})(\text{PB1}) \rightarrow \text{B}$
 $\text{FUN}(\text{C})(\text{PA2}) \rightarrow \text{B}$

Le precedenti definizioni compilano senza provocare errori (con gli opportuni #include e using).
 Per ognuna delle seguenti 12 istruzioni di invocazione della funzione Fun della tabella scrivere chiaramente nel foglio 12 righe con numerazione da 01 a 12 e per ciascuna riga:

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

01:	Fun(&c);
02:	Fun(&d);
03:	Fun(pa1);
04:	Fun(pa2);
05:	Fun(pb1);
06:	Fun(pb2);
07:	Fun<A>(pb1);
08:	Fun<A>(pa2);
09:	Fun(pb1);
10:	Fun<C>(pa2);
11:	Fun<C>(&d);
12:	Fun<D>(pa2);

$\text{FUN}(\text{FUN}(\text{A})(\text{PA2})); \rightarrow \text{A/A}$

\non compila