

Esercizio Funzione

Si assumano le seguenti specifiche riguardanti la libreria Qt.

- (a) Un oggetto della classe `QString` rappresenta una stringa e fornisce un costruttore `QString(const char*)` con il seguente comportamento: `QString(str)` costruisce una `QString` inizializzata con l’array di char `str`.
- (b) `QPaintDevice` è la classe base polimorfa di tutti gli oggetti che possono essere “dipinti”. La classe `QPaintDevice` rende disponibile un metodo `int height() const` con il seguente comportamento: `pd.height()` ritorna l’altezza in pixel del `QPaintDevice pd`.
- (c) `QWidget` è una sottoclasse di `QPaintDevice` i cui oggetti rappresentano delle componenti di una interfaccia grafica Qt. La classe `QWidget` rende disponibile un metodo `bool hasFocus() const` con il seguente comportamento: `w.hasFocus()` ritorna `true` quando la componente `w` detiene il keyboard focus. Inoltre `QWidget` rende disponibile un metodo `void clearFocus()` con il seguente comportamento: `w.clearFocus()` toglie il keyboard focus alla `QWidget w`.
- (d) `QAbstractButton` è derivata direttamente da `QWidget` ed è la classe base astratta dei widget pulsante. La classe `QAbstractButton` rende disponibile un metodo `void setText(const QString&)` con il seguente comportamento: `b.setText(s)` setta l’etichetta testuale del `QAbstractButton b` alla stringa `s`.

Definire una funzione `vector<QWidget> fun(const vector<const QPaintDevice*>&)` tale che in ogni invocazione `fun(v)`:

- (1) per ogni puntatore `p` contenuto nel vector `v`:
 - se `p` punta ad un oggetto che è un `QWidget` con altezza > 50 pixel allora lancia una eccezione di tipo `QString` che rappresenta la stringa “TooBig”;
 - se `p` punta ad un oggetto `obj` che è un `QWidget` con altezza ≤ 50 pixel che detiene il keyboard focus allora toglie il keyboard focus a `obj`;
 - se `p` punta ad un oggetto `obj` che è un `QAbstractButton` allora setta l’etichetta testuale di `obj` alla stringa “Pulsante”.
- (2) l’invocazione `fun(v)` deve ritornare un vector contenente una copia di tutti e soli i `QWidget` che non sono un `QAbstractButton` puntati da un puntatore contenuto nel vector `v`.

```
vector<QWidget> fun(const vector<const QPaintDevice*>& v){
    vector<QWidget> ret;

    for(auto it = v.begin(); it != v.end(); ++it){
        QWidget* q = dynamic_cast<QWidget*>(*it);

        if(q && q->height() > 50){
            throw QString("TooBig");
        }

        if(q && q->height() ≤ 50 && q->hasFocus()){
            q->clearFocus();
        }

        QAbstractButton* qa = dynamic_cast<QAbstractButton*>(*it);

        if(qa){
            qa->setText("Pulsante");
        }

        if(!qa && q){
            ret.push_back(q);
        }
    }
    return ret;
}
```

A() ... F(). $\bar{F} \neq \text{punt } F = \text{new } F()$

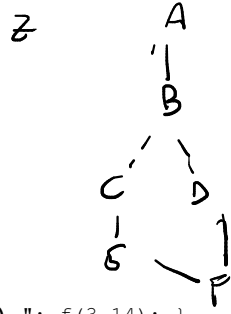
Esercizio Cosa Stampa

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

class A {
public:
    A() {cout << "A() ";}
    ~A() {cout << "~A ";}
};

class B: public A {
public:
    void f(int) {cout << "B::f(int) "; f(3.14);}
    virtual void f(double) {cout << "B::f(double) ";}
    virtual B* f(Z) {cout << "B::f(Z) "; return this;}
    B() {cout << "B() ";}
    ~B() {cout << "~B ";}
};

class C: virtual public B {
public:
    virtual void f(const int&) {cout<< "C::f(const int&) ";}
    virtual C* f(Z) {cout << "C::f(Z) "; return this;}
    C() {cout << "C() ";}
    virtual ~C() {cout << "~C ";}
};
```



```
class D: virtual public B {
public:
    D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
    virtual void f(double) {cout << "D::f(double) ";}
    D() {cout << "D() ";}
    ~D() {cout << "~D ";}
};

class E: public C {
public:
    virtual void f() {cout << "E::f() "; C::f(Z(1));}
    C* f(Z) {cout << "E::f(Z) "; f(); return this;}
    E() {cout << "E() ";}
    E(const E& e) {cout << "Ec ";}
    ~E() {cout << "~E ";}
};

class F: public E, public D {
public:
    void f() const {cout << "F::f() ";}
    F* f(Z) {cout << "F::f(Z) "; return this;}
    void f(double) {cout << "F::f(double) ";}
    F() {cout << "F() ";}
    ~F() {cout << "~F ";}
};
```

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pbl=pd, *pb3=pf; C* pc=pf;

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

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

01: F* puntF = new F;
02: E* puntE = new E(*pe);
03: pb3->f(3);
04: pa->f(1.2);
05: pbl->f(Z(2));
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));
07: static_cast<E*>(pc)->f();
08: pe->f(2);
09: (pc->f(Z(3)))->f(4);
10: (pb3->f(Z(3)))->f(4);
11: delete pb3;
12: delete pe;

Esercizio Cosa Stampa

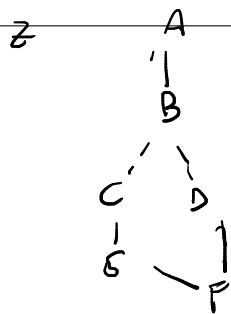
```
class Z {
public: Z(int x) {}
};

class A {
public:
    A() {cout << "A() "; }
    ~A() {cout << "~A "; }
};

class B: public A {
public:
    void f(int) {cout << "B::f(int) "; f(3.14); }
    virtual void f(double) {cout << "B::f(double) "; }
    virtual B* f(Z) {cout << "B::f(Z) "; return this; }
    B() {cout << "B() "; }
    ~B() {cout << "~B "; }
};

class C: virtual public B {
public:
    virtual void f(const int&) {cout<< "C::f(const int&) "; }
    virtual C* f(Z) {cout << "C::f(Z) "; return this; }
    C() {cout << "C() "; }
    virtual ~C() {cout << "~C "; }
};

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pb1=pd, *pb3=pf; C* pc=pf;
```



CPLA
 $5 \times \text{PUNT} 5 = \text{MSW } 5 (+ \text{PE});$

```
class D: virtual public B {
public:
    D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
    virtual void f(double) {cout << "D::f(double) "; }
    D() {cout << "D() "; }
    ~D() {cout << "~D "; }
};

class E: public C {
public:
    virtual void f() {cout << "E::f() "; C::f(Z(1)); }
    C* f(Z) {cout << "E::f(Z) "; f(); return this;}
    E() {cout << "E() "; }
    E(const E& e) {cout << "Ec "; }
    ~E() {cout << "~E "; }
};

class F: public E, public D {
public:
    void f() const {cout << "F::f() "; }
    F* f(Z) {cout << "F::f(Z) "; return this;}
    void f(double) {cout << "F::f(double) "; }
    F() {cout << "F() "; }
    ~F() {cout << "~F "; }
};
```

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

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

01: F* puntF = new F;
02: E* puntE = new E(*pe);
03: pb3->f(3);
04: pa->f(1.2);
05: pb1->f(Z(2));
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));
07: static_cast<E*>(pc)->f();
08: pe->f(2);
09: (pc->f(Z(3)))->f(4);
10: (pb3->f(Z(3)))->f(4);
11: delete pb3;
12: delete pe;

Esercizio Cosa Stampa

06: if (typeid(pb3)==typeid(F)) pb3->f(Z(2));

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

class A {
public:
    A() {cout << "A()" ;}
    ~A() {cout << "~A " ;}
};

class B: public A {
public:
    void f(int) {cout << "B::f(int) "; f(3.14); }
    virtual void f(double) {cout << "B::f(double) ";}
    virtual B* f(Z) {cout << "B::f(Z) "; return this; }
    B() {cout << "B()" ;}
    ~B() {cout << "~B " ;}
};

class C: virtual public B {
public:
    virtual void f(const int&) {cout<< "C::f(const int&) " ;}
    virtual C* f(Z) {cout << "C::f(Z) "; return this;}
    C() {cout << "C()" ;}
    virtual ~C() {cout << "~C " ;}
};
```

```
class D: virtual public B {
public:
    D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
    virtual void f(double) {cout << "D::f(double) ";}
    D() {cout << "D()" ;}
    ~D() {cout << "~D " ;}
};
```

```
class E: public C {
public:
    virtual void f() {cout << "E::f() "; C::f(Z(1));}
    C* f(Z) {cout << "E::f(Z) "; f(); return this;}
    E() {cout << "E()" ;}
    E(const E& e) {cout << "Ec " ;}
    ~E() {cout << "~E " ;}
};
```

```
class F: public E, public D {
public:
    void f() const {cout << "F::f() " ;}
    F* f(Z) {cout << "F::f(Z) "; return this;}
    void f(double) {cout << "F::f(double) " ;}
    F() {cout << "F()" ;}
    ~F() {cout << "~F " ;}
};
```

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pbl=pd, *pb3=pf; C* pc=pf;

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

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

01: F* puntF = new F;
02: E* puntE = new E(*pe);
03: pb3->f(3);
04: pa->f(1.2);
05: pbl->f(Z(2));
06: if (typeid(pb3)==typeid(F)) pb3->f(Z(2));
07: static_cast<E*>(pc)->f();
08: pe->f(2);
09: (pc->f(Z(3)))->f(4);
10: (pb3->f(Z(3)))->f(4);
11: delete pb3;
12: delete pe;

Esercizio Cosa Stampa

A/F

PA → F(1.2);

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

class A {
public:
    A() {cout << "A() "; }
    ~A() {cout << "~A ";}
};

class B: public A {
public:
    void f(int) {cout << "B::f(int) "; f(3.14); }
    virtual void f(double) {cout << "B::f(double) ";}
    virtual B* f(Z) {cout << "B::f(Z) "; return this; }
    B() {cout << "B() "; }
    ~B() {cout << "~B ";}
};

class C: virtual public B {
public:
    virtual void f(const int&) {cout<< "C::f(const int&) ";}
    virtual C* f(Z) {cout << "C::f(Z) "; return this;}
    C() {cout << "C() "; }
    virtual ~C() {cout << "~C ";}
};

class D: virtual public B {
public:
    D* f(Z) {cout << "D::f(Z) "; f(3.14); return this;}
    virtual void f(double) {cout << "D::f(double) ";}
    D() {cout << "D() ";}
    ~D() {cout << "~D ";}
};

class E: public C {
public:
    virtual void f() {cout << "E::f() "; C::f(Z(1));}
    C* f(Z) {cout << "E::f(Z) "; f(); return this;}
    E() {cout << "E() "; }
    E(const E& e) {cout << "Ec ";}
    ~E() {cout << "~E ";}
};

class F: public E, public D {
public:
    void f() const {cout << "F::f() ";}
    F* f(Z) {cout << "F::f(Z) "; return this;}
    void f(double) {cout << "F::f(double) ";}
    F() {cout << "F() "; }
    ~F() {cout << "~F ";}
};

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pbl=pd, *pb3=pf; C* pc=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

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

01: F* puntF = new F;
02: E* puntE = new E(*pe);
03: pb3->f(3);
04: pa->f(1.2);
05: pbl->f(Z(2));
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));
07: static_cast<E*>(pc)->f();
08: pe->f(2);
09: (pc->f(Z(3)))->f(4);
10: (pb3->f(Z(3)))->f(4);
11: delete pb3;
12: delete pe;

PC → (F(Z(3))) → F(4) / F::F(double)

Esercizio Cosa Stampa

```

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

class A {
public:
    A() {cout << "A() "; }
    ~A() {cout << "~A "; }
};

class B: public A {
public:
    void f(int) {cout << "B::f(int) "; f(3.14); }
    virtual void f(double) {cout << "B::f(double) "; }
    virtual B* f(Z) {cout << "B::f(Z) "; return this; }
    B() {cout << "B() "; }
    ~B() {cout << "~B "; }
};

class C: virtual public B {
public:
    virtual void f(const int&) {cout<< "C::f(const int&) "; }
    virtual C* f(Z) {cout << "C::f(Z) "; return this; }
    C() {cout << "C() "; }
    virtual ~C() {cout << "~C "; }
};

class D: virtual public B {
public:
    D* f(Z) {cout << "D::f(Z) "; f(3.14); return this; }
    virtual void f(double) {cout << "D::f(double) "; }
    D() {cout << "D() "; }
    ~D() {cout << "~D "; }
};

class E: public C {
public:
    virtual void f() {cout << "E::f() "; C::f(Z(1)); }
    C* f(Z) {cout << "E::f(Z) "; f(); return this; }
    E() {cout << "E() "; }
    E(const E& e) {cout << "Ec "; }
    ~E() {cout << "~E "; }
};

class F: public E, public D {
public:
    void f() const {cout << "F::f() "; }
    F* f(Z) {cout << "F::f(Z) "; return this; }
    void f(double) {cout << "F::f(double) "; }
    F() {cout << "F() "; }
    ~F() {cout << "~F "; }
};

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pbl=pd, *pb3=pf; C* pc=pf;

```

Handwritten notes:
 - Above class Z: $PC \rightarrow (F(Z(3))) \rightarrow F(4)$
 - Next to class B: $F::F(Z)$
 - Next to class F: $F::F(double)$
 - Circled numbers 1, 2, 3 pointing to specific lines in the code.

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

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

01: F* puntF = new F;
02: E* puntE = new E(*pe);
03: pb3->f(3);
04: pa->f(1.2);
05: pbl->f(Z(2));
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));
07: static_cast<E*>(pc)->f();
08: pe->f(2);
09: (pc->f(Z(3)))->f(4);
10: (pb3->f(Z(3)))->f(4);
11: delete pb3;
12: delete pe;

$(pb3 \rightarrow (f(z(3))) \rightarrow f(4))$

Esercizio Cosa Stampa

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

class A {
public:
    A() {cout << "A() "; }
    ~A() {cout << "~A "; }
};

class B: public A {
public:
    void f(int) {cout << "B::f(int)"; f(3.14); }
    virtual void f(double) {cout << "B::f(double) "; }
    virtual B* f(Z) {cout << "B::f(Z) "; return this; }
    B() {cout << "B() "; }
    ~B() {cout << "~B "; }
};

class C: virtual public B {
public:
    virtual void f(const int&) {cout<< "C::f(const int&) "; }
    virtual C* f(Z) {cout << "C::f(Z) "; return this; }
    C() {cout << "C() "; }
    virtual ~C() {cout << "~C "; }
};

class D: virtual public B {
public:
    D* f(Z) {cout << "D::f(Z) "; f(3.14); return this; }
    virtual void f(double) {cout << "D::f(double) "; }
    D() {cout << "D() "; }
    ~D() {cout << "~D "; }
};

class E: public C {
public:
    virtual void f() {cout << "E::f() "; C::f(Z(1)); }
    C* f(Z) {cout << "E::f(Z) "; f(); return this; }
    E() {cout << "E() "; }
    E(const E& e) {cout << "Ec "; }
    ~E() {cout << "~E "; }
};

class F: public E, public D {
public:
    void f() const {cout << "F::f()"; }
    F* f(Z) {cout << "F::f(Z)"; return this; }
    void f(double) {cout << "F::f(double)"; }
    F() {cout << "F() "; }
    ~F() {cout << "~F "; }
};

A* pa = new F; D* pd = new D; E* pe = new E; F* pf = new F; B *pbl=pd, *pb3=pf; C* pc=pf;
```

Le precedenti definizioni compilano correttamente. Per ognuno dei seguenti 12 statement in tabella con **numerazione da 01 a 12**, scrivere **chiaramente nel foglio 12 risposte con numerazione da 01 a 12** e per ciascuna risposta:

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

01: F* puntF = new F;
02: E* puntE = new E(*pe);
03: pb3->f(3);
04: pa->f(1.2);
05: pbl->f(Z(2));
06: if(typeid(pb3)==typeid(F)) pb3->f(Z(2));
07: static_cast<E*>(pc)->f();
08: pe->f(2);
09: (pc->f(Z(3)))->f(4);
10: (pb3->f(Z(3)))->f(4);
11: delete pb3;
12: delete pe;