Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 15/2/2021

Esercizio Cosa Stampa

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

```
class A {
 public:
  void f(int) {cout << "A::f(int) "; f(true);}
  virtual void f(bool) {cout << "A::f(bool) ";}
  virtual A* f(Z) {cout << "A::f(Z) "; f(2); return this;}
  A() {cout << "A() "; }
};
```

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

```
class C: virtual public A {
 public:
  C* f(Z) {cout << "C::f(Z) "; return this;}
  C() {cout << "C() "; }
};
```

```
class D: virtual public A {
public:
  virtual void f(bool) const {cout << "D::f(bool) ";}
  A* f(Z) {cout << "D::f(Z) "; return this;}
  ~D() {cout << "~D() ";}
  D() {cout << "D() ";}
};
```

```
class E: public C {
 public:
  C* f(Z) {cout << "E::f(Z) "; return this;}
  ~E() {cout << "~E() ";}
  E() {cout << "E() ";}
};
```

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

```
B *pb1 = new F; B *pb2 = new B;
C *pc = new C; E *pe = new E;
A *pa1 = pc, *pa2 = pe, *pa3 = new F;
```

Le precedenti definizioni compilano correttamente.
Per ognuna delle seguenti 12 istruzioni 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 BEHAVIOR** se l'istruzione compila correttamente ma la sua esecuzione provoca un undefined behavior 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: pa3->f(3);                          .....
02: pb1->f(true);                       .....
03: pa2->f(true);                       .....
04: pa1->f(Z(2));                       .....
05: (dynamic_cast<E*>(pa2))->f(Z(2));   .....
06: (dynamic_cast<C*>(pa3))->f(Z(2));   .....
07: pb2->f(3);                          .....
08: (pa2->f(Z(3)))->f(4);               .....
09: (pc->f(Z(3)))->f(4);                .....
10: E* puntE = new F;                   .....
11: delete pa3;                         .....
12: delete pb1;                         .....
```

Esercizio Funzione

Si **assumano** le seguenti specifiche riguardanti la libreria Qt (**ATTENZIONE: non si tratta di codice da definire.**).

- Come noto, `QWidget` è la classe base polimorfa di tutte le classi Gui della libreria Qt. La classe `QWidget` rende disponibile un metodo virtuale `int heightDefault() const` con il seguente comportamento: `w.heightDefault()` ritorna l'altezza di default del widget `w`.
- La classe `QFrame` deriva direttamente e pubblicamente da `QWidget`. La classe `QFrame` rende disponibile un metodo `void setLineWidth(int)` con il seguente comportamento: `f.setLineWidth(z)` imposta la larghezza della cornice del frame `f` al valore `z`.
- La classe `QLabel` deriva direttamente e pubblicamente da `QFrame`.
  - La classe `QLabel` fornisce un overriding del metodo virtuale `QWidget::heightDefault()`.
  - La classe `QLabel` rende disponibile un metodo `void setWordWrap(bool)` con il seguente comportamento: `l.setWordWrap(b)` imposta al valore booleano `b` la proprietà di word-wrapping (andare a capo automaticamente) della label `l`.
- La classe `QSplitter` deriva direttamente e pubblicamente da `QFrame`.
- La classe `QLCDNumber` deriva direttamente e pubblicamente da `QFrame`. La classe `QLCDNumber` rende disponibile un metodo `void setDigitCount(int)` con il seguente comportamento: `lcd.setDigitCount(z)` imposta al valore `z` il numero di cifre dell'intero memorizzato dal lcdNumber `lcd`.

Definire una funzione `fun` di segnatura `list<QFrame*> fun(vector<QWidget*>&)` con il seguente comportamento: in ogni invocazione `fun(v)`,

1. per ogni puntatore `p` elemento del vector `v`:
   - se `*p` è un `QLabel` allora imposta la larghezza della sua cornice al valore 8 ed imposta a `false` la proprietà di word-wrapping della label `*p`;
   - se `*p` è un `QLCDNumber` allora imposta al valore 3 il numero di cifre dell'intero memorizzato dal lcdNumber `*p`.
2. `fun(v)` deve ritornare una lista contenente **tutti e soli** i puntatori `p` non nulli contenuti nel vector `v` che puntano ad un `QFrame` che non è un `QSplitter` e la cui altezza di default è minore di 10.