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.

```
list<QFrame*> fun(vector<QWidget*>& v){
        list<QFrame*> ret;

        for(auto p = v.begin(); p =/= v.end(); ++p){
                // sottotipo di QLabel
                QLabel* ql = dynamic_cast<QLabel*>(*p);

                // 1.a
                if(ql){
                        ql->setLineWidth(8);
                }
                // alternativa = chiama dentro l'if (lunga)
                if(dynamic_cast<QLabel*>(*qp) && dynamic_cast<QLabel*>(*qp)->setWordwrap){}

                // 1.b
                QLCDNumber* qlcd = dynamic_cast<QLCDNumber*>(*p);
                if(qlcd){
                        ql->setDigitCount(3);
                }

                // 2 (ritorno i QFrame non QSplitter)
                if(dynamic_cast<QFrame*>(*p) && !dynamic_cast<QSplitter*>(*p) &&
                p->heigthDefault() < 10){
                        v.push_back(p);
                }
        }

        return ret;
}
```

Siano A, B, C, D, E cinque diverse classi polimorfe.

Si considerino le seguenti definizioni:

```
template<class C, class D>
const C* fun(C& r){
    const C* ptr = &r; return dynamic_cast<D*>(ptr);
}

int main(){
    B b; C c; D d; E e;
    if (fun<A,B>(c) == nullptr) cout << "Bjarne" << endl;
    if (dynamic_cast<C*>(&b) == nullptr) cout << "Stroustrup" << endl;
    const A* p = fun<D,B>(d);
    const D* q = fun<E,B>(e);
    fun<C,D>(d);
}
```

Definire una unica gerarchia di classi che soddisfi **tutti** i seguenti requisiti:

1. Una classe base polimorfa astratta `Component` alla radice della gerarchia, che possiede un metodo virtuale puro `render()` e un metodo virtuale `getInfo()` che ritorna una stringa.
2. Una classe astratta `Container` derivata da `Component` che aggiunge un metodo virtuale puro `add(Component*)` ed un metodo non virtuale `remove()`.
3. Una classe `Panel` derivata da `Container` che implementa concretamente i metodi virtuali puri ereditati. Questa classe aggiunge un metodo specifico `setLayout(string)`.
4. Una classe `Button` derivata da `Component` che implementa concretamente i metodi virtuali puri ereditati e aggiunge un metodo specifico `click()`.
5. Una classe `ImageComponent` che non permette la costruzione pubblica dei suoi oggetti, ma solamente la costruzione di oggetti che siano sottooggetti. La classe possiede un metodo virtuale `resize()`.
6. Una classe `ImageButton` derivata da `Button` tramite ereditarietà pubblica e da `ImageComponent` tramite ereditarietà privata. La classe ha un metodo aggiuntivo `setImage(string)`.
7. Una classe `Window` definita mediante derivazione multipla che eredita sia da `Panel` che da `ImageComponent` con ereditarietà virtuale. La classe `Window` ridefinisce pubblicamente l'operatore di assegnazione con comportamento identico a quello dell'assegnazione standard.

```
// 1
class Component{
public:

virtual void render() = 0;
virtual std::string getInfo();
virtual ~Component() {}
};

// 2

class Container: public
Component{
public:

virtual void add(Component*)=0;
void remove() const;
};
```

3. Una classe `Panel` derivata da `Container` che implementa concretamente i metodi virtuali puri ereditati. Questa classe aggiunge un metodo specifico `setLayout(string)`.

```
class Panel: public Container{
public:
        virtual void add(Component* c){

        }
        virtual void render(){
                .
        }
        void setLayout(std::string) const;
};
```

4. Una classe `Button` derivata da `Component` che implementa concretamente i metodi virtuali puri ereditati e aggiunge un metodo specifico `click()`.
5. Una classe `ImageComponent` che non permette la costruzione pubblica dei suoi oggetti, ma solamente la costruzione di oggetti che siano sottooggetti. La classe possiede un metodo virtuale `resize()`.
6. Una classe `ImageButton` derivata da `Button` tramite ereditarietà pubblica e da `ImageComponent` tramite ereditarietà privata. La classe ha un metodo aggiuntivo `setImage(string)`.
7. Una classe `Window` definita mediante derivazione multipla che eredita sia da `Panel` che da `ImageComponent` con ereditarietà virtuale. La classe `Window` ridefinisce pubblicamente l'operatore di assegnazione con comportamento identico a quello dell'assegnazione standard.

```
class Button: public Component{
        public:
                virtual void render();
                void click() const;
};

class ImageComponent{
        private:
                int x;
        protected:
                ImageComponent(int x1): x(x1) {}
                virtual void resize() const;
};

class ImageButton: public Button, private ImageComponent{
        public:
                void setImage(std::string) const;
};

class Window: virtual public Panel, virtual public ImageComponent{
        // assegnazione standard

        Window& operator=(const Window& w){
                // if(this ≠ w) - se non ereditarietà

                // standard
                Panel::operator=(w);
                ImageComponent::operator(w);

                // assegnazione campi → campo = rif.campo;

                return *this;
        }
};
```

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

class B: public A {};
class C: public A {};
class D: public B, public C {};
class E: public D {};
class F: public B {};
class G: public F {};

std::string H(A* pa, B* pb) {
    if (dynamic_cast<F*>(pa)) {
        if (dynamic_cast<G*>(pa))
            return "7";
        return "3";
    }

    if (dynamic_cast<D*>(pa)) {
        if (dynamic_cast<E*>(pa))
            return "1";
        return "6";
    }

    if (dynamic_cast<C*>(pb))
        return "4";

    if (typeid(*pa) == typeid(*pb))
        return "8";

    return "0";
}
```

Si consideri inoltre il seguente `main()` incompleto:

```cpp
int main() {
    A a; B b; C c; D d; E e; F f; G g;

    cout << H(..., ...) << H(..., ...) << H(..., ...)
         << H(..., ...) << H(..., ...) << H(..., ...)
         << H(..., ...) << H(..., ...);
}
```

Definire opportunamente le chiamate alla funzione `H()` in questo `main()` usando gli oggetti locali `a`, `b`, `c`, `d`, `e`, `f`, `g` in modo tale che:

1. Il `main()` compili correttamente ed esegua senza provocare errori a run-time
2. Le chiamate ad `H()` siano tutte diverse tra loro
3. L'esecuzione produca in output esattamente la stampa "31415926"