

Esercizio Funzione

<pre>class A { public: virtual A* f() const =0; }; class D: public B {};</pre>	<pre>class B: public A {}; class E: public B { public: A* f() const {return new E();} };</pre>	<pre>class C: public B { public: B* f() const {return new C();} }; class F: public C, public D, public E { public: D* f() const {return new F();} };</pre>
---	---	---

Queste definizioni compilano correttamente. Definire una funzione

```
list<const D *const> fun(const vector<const B*>&)
```

con il seguente comportamento: in ogni invocazione `fun(v)`, per tutti i puntatori `q` contenuti nel vector `v`:

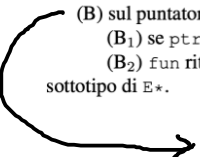
(A) se `q` non è nullo ed ha un tipo dinamico esattamente uguale a `C` allora `q` deve essere rimosso da `v`;

(A₁) se il numero `N` di puntatori rimossi dal vector `v` è maggiore di 2 allora viene sollevata una eccezione di tipo `C`.

(B) sul puntatore `q` non nullo deve essere invocata la funzione virtuale pura `A* A::f()` che ritorna un puntatore che indichiamo qui con `ptr`;

(B₁) se `ptr` è nullo allora viene sollevata una eccezione `std::string("nullptr")`;

(B₂) `fun` ritorna la lista di tutti e soli questi puntatori `ptr` che: non sono nulli ed hanno un tipo dinamico che è sottotipo di `D*` e non è un sottotipo di `E*`.

 `A* ptr = (*q)->f();`

```
list<const D* const> fun(const vector<const B*> & v){
    list<const D* const> result;
    int cont = 0;

    for(std::vector<const B*>::const_iterator q = v.begin(); q != v.end(); ++q){
        if(q != nullptr && typeid(C) == typeid(**q)){
            B* b = const_cast<B*>(*q);
            q = v.erase(b);
            delete q;

            cont++;
            if(cont > 2) throw C();
        }

        A* ptr = (*q)->f();
        if(ptr == nullptr)
            throw std::string("nullptr");
        else{
            if(dynamic_cast<D*>(ptr) && !dynamic_cast<E*>(ptr))
                result.push_back(dynamic_cast<const D* const>(ptr));
        }
        // if(dynamic_cast<const D *const>(ptr) && !dynamic_cast<const E *const>(ptr))
    }

    return result;
}

// Cancellazione con riposizionamento manuale - occhio ad "erase" (non esiste const_erase)
// Soluzioni MEGA vecchie (pre 2019/18)
vector<const B*>::const_iterator q = v.begin();

while(it != v.end()){
    if(q != nullptr && typeid(C) == typeid(**q)){
        B* b = const_cast<B*>(*q);
        q = v.erase(b);
        delete q;
        cont++;
    }
    else{
        it--;
    }
    ++it;
}
```